2

AFWAL-TR-81-3013

SUPPLEMENTARY STUDIES ON THE SENSITIVITY OF OPTIMIZED STRUCTURES

PAUL S. JENSEN
W. A. LODEN

APPLIED MECHANICS LABORATORY
LOCKHEED PALO ALTO RESEARCH LABORATORY
3251 HANOVER STREET
PALO ALTO, CALIFORNIA 94304

MARCH 1981

TECHNICAL REPORT AFWAL-81-3013
Final Report for Period 14 November 1979 - 29 September 1980

FLIGHT DYNAMICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

Narendra S Khot

NARENDRA S. KHOT,
Project Engineer

Fred A. Picchioni

FREDERICK A. PICCHIONI, Lt Col, USAF
Chief, Analysis & Optimization Branch

FOR THE COMMANDER

for [illegible]

RALPH L. KUSTER, JR., Col, USAF
Chief, Structures & Dynamics Div.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR-81-3013 / 2. GOVT ACCESSION NO. [illegible]	3. RECIPIENT'S CATALOG NUMBER [illegible]
4. TITLE (and Subtitle) SUPPLEMENTARY STUDY ON THE SENSITIVITY OF OPTIMIZED STRUCTURES	5. TYPE OF REPORT & PERIOD COVERED Final Report for Period 14 Nov 79 - 29 Sep 80
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Paul S. Jensen, W. A. Loden	8. CONTRACT OR GRANT NUMBER(s) F33615-76-C-3105 Mod P00007
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lockheed Palo Alto Research Laboratory 3251 Hanover St. (52-33/205) Palo Alto, California 94304 /10118	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2307N102
11. CONTROLLING OFFICE NAME AND ADDRESS Flight Dynamics Laboratory (AFWAL/FIBRA) Air Force Wright Aeronautical Laboratories (AFSC) Wright-Patterson AFB, Ohio 45433	12. REPORT DATE March 1981
	13. NUMBER OF PAGES 78
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Sparse Matrices, Structural Analysis, Optimization, Virtual Memory

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Reports of three related studies germane to structural optimization are included under one heading. The first describes virtual memory simulator suitable for management of large quantities of numerical data such as required for sparse matrix manipulation. The second report describes two sparse matrix processors suitable for the large equation systems arising in structural analysis and provides comparative results. The last report describes a study of two optimization algorithms in the context of structural optimization. A number of test results for parameter studies and a general comparison of the two algorithms are given.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED [illegible]

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

FOREWORD

This report was prepared by Lockheed Missiles and Space Company, Inc., Palo Alto Research Laboratories, 3251 Hanover Street, Palo Alto, California, in partial fulfillment of the requirements under Contract F33615-76-C-3105. The effort was initiated under Project 2307, "Research in Flight Vehicle Structures," Task 2307N102, "Research in the Behavior of Metallic and Composite Components of Air Frame Structures." The project monitor for the contract was Dr. Narendra S. Khot of the Structures and Dynamics Division (AFWAL/FIBRA).

The technical work under the contract was performed during the period June 1976 through October 1980. Review report was submitted in October 1980 and the final report in March 1981.

The other reports published under this contract are "Imperfection Sensitivity of Optimized Structures," (AFWAL-TR-80-3128), "Numerical Procedure for Analysis of Structural Shells," (AFWAL-TR-80-3129), "Panel Optimization with Integrated Software (POIS)," (AFWAL-TR-80-3073, Vol I and II), "Design of Composite Material Structures for Buckling, An Evaluation of State-of-the-Art," (AFWAL-TR-81-3102).

TABLE OF CONTENTS

INTRODUCTION

Paul S. Jensen

The work carried out under this supplemental effort to the general study on the sensitivity of optimized structures was divided into three tasks:

1. Enhance the virtual memory simulator program VMSYST that supports the sparse matrix processor. Specifically, improve its efficiency for handling very large records and reduce its overhead costs for small records.

2. Comparatively evaluate the profile and generalized wave front algorithms for processing large sparse matrix problems arising in structural analysis.

3. Implement a local, constrained optimization algorithm based on the method of augmented Lagrangians and evaluate it for the optimization of stiffened panels.

These three tasks are all supportive to the general problem of structural design and optimization. They are sufficiently different, however, to permit treatment as three autonomous efforts. Detailed discussions of the tasks are provided in the separate articles in this report. An overview of the project and summaries of the results of the tasks are provided here.

Structural optimization is carried out in a sequence of steps. For each step, the following major computational tasks are carried out:

1. The optimizer program produces a new structural configuration based on the information available from the previous steps. Presumably, this new configuration is an improvement to the previous ones in the sense that it meets the constraint requirements and improves the objective function, e.g., reduces the overall weight. Often this task is done by a human analyst.

2. The structural analysis program analyzes the new configuration produced by the optimizer with regard to the constraint and objective functions.

It is clear that each step tends to be rather costly for complex structures because it requires a detailed analysis of a structural configuration. Consequently, there is a strong incentive to minimize both the total number of steps required and the cost of each step. The number of optimization steps is controlled by the "optimizer" (optimization program) and the cost of each step is determined by the "structural analyzer". The first two tasks of this project relate to improvements in the structural analyzer and the third task concerns the optimizer.

VIRTUAL MEMORY SIMULATOR

Although a geat deal of progress in general sparse matrix processing has been achieved over the past decade, one fundamental difficulty has inhibited wide acceptance of these new results for structural analysis applications. That difficulty centers on the management of the vast quantities of data that are generated. The traditional profile methods used for structural analysis have a data structure that permits simple management techniques relative to those required for the newer general algorithms. The most attractive data management approach for the new algorithms is based on the concept of word addressable virtual memory. Since many of the computers used for structrual analysis do not have virtual memory hardware, the only option available is to simulate virtual memory by software.

Prior to this study, a comprehensive system of software modules was developed to efficiently manage large quantities of data in the spirit of virtual memory systems. In this study, extensive enhancements to that system were made including:

1. Reduced overhead costs by delayed validity checking,

2. Direct (unbuffered) transfer of very large records and

3. Dynamically variable operational characteristics that, among other things, permits release of buffer core area when not needed.

SPARSE MATRIX PROCESSING

The profile (variable band) sparse matrix processing techniques traditionally used for structural analysis have proven reasonably satisfactory, especially in light of efficient preprocessing algorithms that automatically produce good equation oderings that yield tightly banded matrices. The objective of this task was to compare a good available profile algorithm (including preprocessor) with a new algorithm based on a generalized wave front technique that was developed under previous Air Force support.

Previous work with that algorithm (called SPSYST) revealed several aspects that needed further work. Unfortunately, that work, which is being carried out by Dr. J.K.Reid at AERE Harwell in England, was not completed in time for this study. Consequentl,, the tests were made with an old version of SPSYST.

Besides completing a number of representative tests, a significant accomplishment of this task was the development of a convenient host program, called SPARTA (SPARse Test Algorithm). It is designed to facilitate tests of this nature and, more generally, to interface SPSYST with a variety of general purpose structural analysis programs. It is anticipated that SPARTA will be of considerable use in future research studies and production applications.

Five structural problems that have been analyzed previously by engineers at Lockheed Misseles and Space Co. (LMSC) were used for this study. The profile method produced the less costly results for the relatively simple problems (as might be expected) and SPSYST did better for certain complex problems. Generally, it appears that the new algorithm is quite promising.

OPTIMIZATION

The general structural optimization problem has a number of characteristics that make it particularly challenging. As briefly mentioned earlier, the costs of obtaining pertinant information from the structural analyzer are generally quite large. Consequently, the optimizer must use the information it gets very effectively. An even worse characteristic is that the constraint functions (e.g., buckling load) are occasionally not differentiable with respect to the design variables. In any case, important gradients of these functions are not available except by finite difference approximation.

Because of the limited funding available for this study, the only tests run involved an optimizer that was developed and partially implemented previously. That optimizer, called ALMIN (Adaptive Lagrangian MINimizer), incorporates a number of recent innovations in optimization theory and appeared to have considerable promise for the structural optimization problem. Basically, the objective function is augmented with penalty terms representing the constraint functions. These terms are small when the constraints are satisfied and grow as the constraints become violated. The balance between the penalty terms and the objective function is established by weighting factors applied to the penalty terms. A robust method for determining the weighting factors has not yet been developed for ALMIN.

An implementation of ALMIN was completed and a variety of test problems were run. Although operational, the new algorithm is not suitable for production applications in its current form. A number of critical decision processes are presently carried out using ad hoc heuristics that are neither reliable nor always efficient. Examples of these decision processes are: (1) When to reinitialize the approximate Hessian matrix, (2) What technique to use for increasing the penalty function weights and (3) How to approximate the gradient functions.

Our general feeling is that a number of avenues for considerably improving upon the present technique exist and, because of the substantial analysis costs involved, should be pursued. Some specific areas of interest are:

1. Careful attention to the finite difference techniques used to approximate the gradients of the constraint and objective functions must be given,

2. Innovative utilization of the information available from the structural analyzer (that is not typically used by optimizers) must be found. An example of such information is the eigenvector associated with the buckling load (an eigenvlaue).

3. Incorporation of recent ideas that exploit the properties of the trajectories of penalty and barrier function methods to overcome the problems associated with the penalty weights.

A VIRTUAL MEMORY SIMULATOR

Paul S. Jensen

ABSTRACT

A software package called VMSYST for simulating virtual memory on a variety of computing systems is described. It utilizes the popular LRU (Least Recently Used) paging policy, with variable page and page buffer sizes. Except for the input/output routines, the package is written in standard FORTRAN 66 for transportability. A gauche FORTRAN version of the input/output routines is included for simple testing but is not recommended for production work.

VMSYST was designed to support a sparse matrix package based on a generalized frontal scheme. However, it has gained popularity for use in other applications such as text library management and as a general purpose local data manager for "number crunching" programs. It is currently operational on CDC 6000 and CYBER systems, UNIVAC 1100 series computers and the VAX 11/780.

TABLE OF CONTENTS

1. INTRODUCTION

One of the biggest thorns in developing programs to operate on massive quantities of data (text or numerical) is data management. The fact that FORTRAN has provided a reasonably effective input/output capability has undoubtedly been an important factor in its survival over the years in the face of important competitors such as ALGOL. Never-the-less, the FORTRAN input/output system has traditionally been serial oriented and has proven inadequate for serious computational work. The new FORTRAN 77 standard has substantially improved the input/output capabilities inherent to the language and future standards may one day completely obviate the need for utilities such as the system discussed here. For the present, however, versatile subroutine packages for simplifying data management fill an important need in production computation.

There is a wealth of literature on virtual memory systems describing a broad base of experience gained over the past two decades. Good general discussions may be found in [1,3 and 8]. Some rather discouraging results have been reported in [3 and 7] for example, and encouraging results have appeared in [5,6 and 8]. Our experience indicates that the convenience of virtual memory is an extremely attractive feature and that reasonable efficiency can be achieved for a fairly broad range of problems. Some (variations in page size, resident set size, etc.) is advantageous for many applications but is not essential.

The system described here is fairly typical except for one important innovation germane to large records. When the size of a record is several times the size of a basic page, much of the record is transferred directly instead of through the page buffer. There are other innovations germane to numerical processes such as limited arithmetic operations directly in the page buffer, but these are primarily convenience features. Unfortunately, some jargon needs to be introduced in order to facilitate the discussion. New terms are defined in the text and again in the Appendix along with some additional interesting terms. Most of the jargon has been taken from various references and should be fairly standard.

2. DESIGN HIGHLIGHTS

The unit of information (datum) managed by VMSYST is the word. The size of a word, unfortunately, varies among computers and requires specification for each computer. This and other specifications pertaining to the nature of the data in "virtual memory" is retained with the data in a special virtual header. Physically, the header is retained in the first 25 words of the storage file holding the data. Thus, the header can usually be contained in the first sector of a random access disc storage device. When VMSYST is provided a file that is alleged to hold the contents of a VM (virtual memory), it immediately reads the header and adjusts various internal parameters accordingly. When VMSYST closes a virtual file, it updates the header to reflect the most recent activity. This and certain other processes prevent the use of serial storage media for direct virtual storage.

Except for the header, all data retained on a VM (virtual memory) file appears to an application program as <u>word addressable</u> information. The number of storage locations available depends upon the file size. The sequence of allowed addresses (usually starting with 1) is called the <u>virtual address space</u>. For management purposes, it is partitioned into fixed-size <u>pages</u>. When a relatively small <u>segment</u> of data is required, the page's holding it are moved to the <u>resident set</u> (if they are not there already) and the segment is copied out from there. The resident set occupies a user specified region in common called the <u>page buffer</u>. When a large segment is required, each page holding an end of the segment is paged in through the resident set as above (unless it holds segment data exclusively, i.e., the segment is <u>page aligned</u>). The rest of the segment, however, is transferred directly to (or from) mass storage.

There are two nearly identical versions of VMSYST. The B version holds the resident set in FORTRAN blank common and the L version holds it in labelled common /VMBUFT/. The position of the resident set is the only difference between the two versions. For either version, a user may adjust the size and/or position of the page buffer in common from time to time during the execution of his program in order to optimize the efficiency of the overall program. He normally specifies the starting location, page size and available buffer space and VMSYST determines how many active pages can be accomodated and how much memory space is actually required. The determination of these parameters is nontrivial because the page buffer is used to hold a <u>page table</u> as well as the resident set.

Some special operations such as scaling (multiplying by a constant) a segment in virtual memory containing numerical data are also provided. The arithmetic specified by such operations is carried out directly in the page buffer area of main storage. Consequently, the entire segment is paged into the resident set for such operations regardless of its size.

3. VIRTUAL MEMORY OPERATIONS

The virtual memory operations are presented here as subroutine names (more precisely, subprogram entry points). They are normally called from a FORTRAN driver or application program. The general format for these operations is:

```
VMxxxx(ARG1,ARG2,ARG3,ARG4),
```

where xxxx specifies a particular operation as discussed below.

There are three classes of virtual memory operations, viz.: TRANSFER, ARITHMETIC and ADMINISTRATIVE. The transfer operations are the easiest to understand and the most frequently used. Three basic transfer operations are defined, viz.: GET, PUT and COPY. There are also three arithmetic operations, viz.: SET, ADD and SCL, that are a bit peripheral to "normal" I/O operations but provide a convenient mechanism for doing limited arithmetic right in the VM buffer area of main storage.

The administrative operations facilitate file management (open, close, etc.), resident set size adjustment, parameter value adjustment and the display of certain operational statistics. They are the most complicated, least interesting and of the greatest number.

3.1 TRANSFER OPERATIONS

The transfer operations that move data between mass storage and main (RAM) memory are:

VMGETR	- Get real data from mass storage
VMGETI	- Get integer data from mass storage
VMPUTR	- Put real data in mass storage
VMPUTI	- Put integer data in mass storage

These are in the form of FORTRAN subroutines called in the following format:

```
CALL VMGETR(R,N,F,L)
```

where R is an array of dimension N in main memory (RAM),
N is the number of words to be transferred,
F is the mass storage file holding the VM data and
L is the location of the data in virtual memory.

There is one transfer operation for copying data from the mass storage of one virtual memory to that of another. It has the form

```
CALL VMCOPY(FFROM,LFROM, N, FTO,LTO, R,NR)
```

where FFROM and LFROM	are the file and virtual location numbers for the source virtual memory,
N	- is the number of words to be copied,
FTO and LTO	- are the file and virtual location numbers for the destination virtual memory,
R	- is an array in RAM for holding temporary data (It should be as large as possible),
NR	- is the size of R.

Of course, data must have previously been put in (FFROM, LFROM) but it does not matter about (FTO, LTO).

3.2 ARITHMETIC OPERATIONS

The arithmetic operations that facilitate a limited amount of computation in the VM page buffers are:

VMSETR(C,N,F,L)	- Set N values in VM to constant C,
VMSCLR(C,N,F,L)	- Scale N values in VM by constant C and
VMADDR(C,R,N,F,L)	- Add C*R to N values in VM, where R is a real vector of length N.

Following the conventions for this system, the final R in each name indicates real arithmetic and arguments C and R must represent real data. As with the transfer operations, arguments F and L designate the VM mass storage file and virtual location of the N data items in VM. The results of the arithmetic operations are always left in VM.

3.3 ADMINISTRATIVE OPERATIONS

Unfortunately, there are a number of administrative operations that must be included to establish VM files, initialize the system, handle error (hardware of course) conditions and communicate parameter settings and operational statistics. A summary of these operations follows:

VMDACC	- Access a virtual file using default specifications
VMDEST	- Establish a virtual file using default specs.
VMINIT	- Initialize the VMSYST operational parameters
VMOPEN	- Open a VM mass storage file (old or new)
VMCLOS	- Close a VM mass storage file
VMACC	- Change type of file access (GET, PUT or BOTH)
VMSTAT	- Display key operational statistics
VMHDR	- Fetch and (optionally) display the VM header
VMINQ	- Inquiry to obtain general parameter information
VMFINQ	- Inquiry to obtain file parameter information
VMPINQ	- Inquiry to obtain page table information
VMMOD	- Modify the operating characteristics (page size, buffer size, etc.)
VMERRH	- Define I/O error handling procedure

3.3.1 INITIALIZATION

The simplest way to establish or gain access to a virtual file is to use VMDEST or VMDACC, both of which prompt the user for an external file name, and then initialize and open the file (see VMINIT and VMOPEN below). VMDEST also prompts the user for a title (up to 48 characters) to be inserted in the virtual file header. Both routines have one argument FILNO, e.g.,

```
    VMDEST(FILNO)
```

which is the integer file reference number returned for use in subsequent VM operations.

In sophisticated applications, it is often desireable to exercise greater control over the initialization and file opening processes. In such cases, the first operation must always be VMINIT. It is important that it is first and that it is executed only once. The initialization routine assumes that the calling program has established a page buffer area in FORTRAN common (blank common for version B or labeled common /VMBUFT/ for version L). The typical form of the buffer statement in the users calling program is

```
     COMMON /VMBUFT/ VMBBBB(size)
```

where size is chosen to be the default *(See Appendix)* or that value given in LIST(3) below. The name VMBBBB can be anything that does not conflict with other variable names used in the calling program. In addition, two symbiont definitions must be established in labelled common

```
     COMMON /NITNOT/ NIT,NOT
```

where NIT is the input symbiont, usually fortran unit 5, and NOT is the output symbiont, usually fortran unit 6.

The simplest (default) form of the initialization operation is

```
     VMDFLT.
```

VMDFLT establishes a default page buffer in labled common, sets symbiont parameters and initializes VMSYST by calling VMINIT(0).

For flexibility, however, a user may specify up to 9 initialization parameters in order to satisfy particular requirements by using the form

```
     VMINIT(LIST)
```

where LIST is an integer list of initialization parameters. In this case, the calling program must establish the page buffer and the symbiont parameters. The items in LIST are interpreted as follows:

1. Number of LIST items provided. If LIST(1) < 2, internal default parameters are used.

2. Index of the first word of the page buffer in the buffer common area (default = 1)

3. Length of the buffer area (beyond index LIST(2))

4. VM page size

5. Page capacity of buffer area

6. If LIST(6)=0, make reasonable adjustments if innocuous inconsistencies exist in the LIST data, otherwise abort if an inconsistency is found.

7. Temporary mass storage unit number to be used for holding extensive operational statistics. If this information is not required (normal case) set LIST(7) = 0.

8. Threshold length (in pages) of records for which the buffered mode of data transmission will be used. Portions of longer records will be transmitted directly when possible.

9. Maximum number of VM files to be opened for this application (default = 8).

10. Number of rows in the page table.

Specification of items 8-10 is not recommended, i.e., LIST(1) should be < 8. Any of the above items may be set to 0 in order to cause the internal default value to be used. The actual parameter values used for the initialization process are returned in array LIST for the users edification.

3.3.2 FILE OPERATIONS

The second VM operation to be used in any application must be the file open operation VMOPEN. It is repeated once for every VM file (max. 8) to be opened and precedes any other operations on the file. Corresponding to each VMOPEN operation there must be a VMCLOS operation which is the last operation on the file. However, a closed file may be re-opened if necessary but it must ultimately be closed.

The file operations are the least portable of the system because they involve peculiarities of the host computer. The objective in VMSYST has been to implement those peculiarities in a relatively small I/O program system written in a suitable language (assembly language in some cases). An overview of this system is provided in Section 5. Files are established by the following three administrative operations that utilize the general I/O system:

```
VMOPEN(NAME,NO,PARAMS)
VMCLOS(NO)
VMACC (NO,TYPE)
```

where: NAME is an alpha-numeric file name. See Section 5.1 or [2] for the format of NAME. If it is a string of 1 or more blanks, a default name will be generated.

NO is a file number assigned by VMOPEN. It is used for all subsequent references to the file. The user can affect the internal unit number used by presetting NO as follows:

NO > 0	Use FORTRAN unit NO
NO = 0	Use internal default (18,17,...)
NO < 0	Set LDI =-NO (See Section 5.1)

In all cases, the final value of NO is not necessarily related to the FORTRAN unit number.

PARAMS is an integer vector of descriptive file parameters interpreted as follows (also see Section 5):

1 Equipment type
Tape .... -2 or -1,
Disc .... 0, 1 or 2
Large core .. 3

2 Permanency
Temporary .... 0 or 1,
Existing file ..... 2,3 or 4,
New file .... 5,6, ..., 12

3 Capacity in words (if zero, use internal default)

4 PRU (physical record unit) size in words. If PARAMS(4)=0, the physical sector size of the storage device will be used. The PRU size must be a multiple of the sector size.

5 Type of access 0 - BOTH get and put,
1 - GET only or,

	2 - PUT only
6	No. of words in the title (max. 12)
7,8,...	Title for file header (4 char/word, right fill)
TYPE	is the type of access (equivalent to PARAMS(5))

A typical example of VMOPEN for a computer with a 4 character word length would have NAME dimensioned 3 and set to

```
NAME = ('TYPICAL*VMF ') or (4HTYPI,4HCAL*,4HVMF )
```

Integer vector PARAMS must be dimensioned 6 and could be set to

```
PARAMS = (0, 0, 0, 0, 0, 0)
```

in order to establish a new, temporary disc file of default capacity and PRU size with access type BOTH. The internal I/O routine attempts to determine the PRU size by inquiry to the system monitor program. If this inquiry capability is not available on a particular computer system, the user must provide the PRU size for new files. If an existing file is being referenced, items PARAMS(3) and PARAMS(4) will be replaced by the values found on the VM file header.

Once all necessary information has been placed on a VM file (via VMPUTx), it is often helpful to protect the file from being written on accidentally by the access operation

```
VMACC(FILE,1).
```

Of course, write access can always be restored by the access operation with second argument of 0. Write only access (argument 2) is seldom used.

3.3.3 INQUIRY OPERATIONS

There are five operations provided for obtaining internal information maintained by VMSYST. For simple applications of VMSYST, these operations may be ignored for they have no effect other than to provide information. They have the following form:

VMSTAT	- Print operational statistics on FORTRAN unit 6
VMHDR(FNO,HDR)	- Return file header information
VMINQ(INFO,N)	- Return operational parameters
VMFINQ(FNO,FINFO,NF)	- Return file information
VMPINQ(PNO,PINFO,NP)	- Return page information

where INFO is a vector of length N in which the same information discussed in Section 3.3.1 for

	argument LIST of VMINIT is returned,
FNO	is the VM file number,
HDR	is a 25 word integer array for header data
FINFO	is a vector of length NF in which the file information is returned,
PNO	is the buffer page number,
PINFO	is a vector of length NP in which the specified column of the page table is returned.

In addition, there is a separate, executable program called VMCHK that facilitates checking and scanning the contents of a previously established virtual file. VMCHK is a conversational program that provides the user with a number of simple commands for displaying the header, the extent (number of virtual locations in which data has been stored), and selected contents in a variety of formats.

VMSTAT

The most commonly used inquiry operation is VMSTAT. It may be invoked at various points in an application program in order to display paging and I/O information. The form of the output (FORTRAN unit 6) is illustrated in the following example.

```
                 ---------------------------
                   SUMMARY OF I-O ACTIVITY
                 ---------------------------

 INPUT     OUTPUT    TOTAL       INPUT     OUTPUT    TOTAL
ACCESSES  ACCESSES  ACCESSES     VOLUME    VOLUME    VOLUME
   38        29        67         2133      1827      3960   VIRTUAL
   12         9        21         1421      1249      2670   PHYSICAL

                     PAGING ACTIVITY

   MISSES    HITS    AV SEARCH  AV SEARCH   PAGES  BUFFER
                      PER MISS   PER HIT            SIZE
      17       43       0.54        1.02       20  17920
   ---------------------------------------------------------
```

The information is reasonably self explanatory. A page HIT or MISS means that the referenced page was resident or nonresident. The search refers to how many buffer pages had to be checked in order to find a desired page or determine that it was not resident. Often no search is required due to the classification scheme discussed in the Section 4.

VMHDR

If the negative value of the file number, i.e., CALL VMHDR(-FNO,HDR), is specified, then VMHDR will display the header information in a reasonably attractive fashion as well as providing the actual data in array HDR. The actual arrangement of the data in HDR is subject to change, and is not appropriate to describe here (this information is documented in the subroutine). It contains a title for the file, the date and time it was established, the next free virtual and physical address, the file capacity, the computing machine on which it was established, and some other uninteresting data.

VMFINQ and VMPINQ

VMFINQ and VMPINQ are only needed in very unusual circumstances and are of little concern to the average user. Thus, the reader is invited to skip this subsection unless he has masochistic tendencies.

Referring to the operation formats above, only the amount (NF or NP) of information requested is returned, up to the that available. The value of NF or NP is adjusted to the amount available if it is initially larger in magnitude. NF or NP may be set negative in order to have the requested information printed out on FORTRAN unit 6 by VMFINQ or VMPINQ. The specific items of information returned by VMFINQ are:

1	Number of PRU'S (sectors) per virtual page
2	Current file position (PRU'S)
3	Next free position (PRU'S)
4	File capacity (words)
5	PRU size (words)
6	Access: 1 - Read, 2 - Write, 0 - Both
7	Logical device index (See Section 5)
8	Next free virtual address (words)
9-12	File name (4 word character string)

The specific items of information returned by VMPINQ are:

1	Index of next older buffer page of this class
2	Index of youngest buffer page of class PNO
3	64*(Virtual address) + (File no.) of this page
4	(Alter)*(File PRU address of this page), where Alter=-1 if resident page contents match the mass storage copy =+1 otherwise
5	Index of next younger buffer page of this class
6	Indicator of when this page was last referenced

3.3.4 MODIFICATION AND ERROR HANDLING

The operations discussed in this section are of even less interest to the casual user than the inquiry operations of the previous section. They are included only for the benefit of ardent enthusiasts that wish to do fancy stuff. One such voodoo operation is dynamically changing the operational characteristics of VMSYST occasionally in order to cut computing costs by a few cents (or dollars in some cases). For example, if an application can be partitioned in phases wherein some are I/O bound and some are compute bound, then it may be worthwhile to use a large page buffer during the I/O bound phases and to shrink it (freeing up valuable memory space) during compute bound phases. This type of dynamic modification is accomplished by operation

```
VMMOD(LIST)
```

where array LIST plays precisely the same role as it did for the VMINIT operation discussed in Section 3.3.1.

In the event of some unexpected I/O error, the I/O management system used by VMSYST indicates what the problem is and then calls the subroutine

```
VMERRH(SOURCE,TYPE)
```

where SOURCE is the name (six or fewer characters) of the VMSYST subroutine last activated and
TYPE is the (integer) error type interpreted as follows:

1	Device index out of range
2	Device not declared
3	Device not attached (activated)
4	Non-existent file or record index
5	Mass storage address out of range
6	Zero or negative record size
7	Attempted read from an unreadable device
8	Device overflow (exceeded capacity)
9	Attempted write on an unwritable device
10	Weird error, check STATUS word
11	Requested facility is unavailable
12	A stupid request was made, e.g., free an undefined file

At the time of this writing, VMERRH simply prints the arguments (SOURCE, TYPE) and the VMSYST file table (see VMOPEN) and aborts the run. A bold user may write his own error handling version of VMERRH if that is his forte.

4. PAGING POLICY

As mentioned earlier, the basic LRU (Least Recently Used) paging policy is used in VMSYST. In this section we discuss the implementation of that policy in some detail.

4.1 PAGE TABLE

The page buffer provides storage for a number m (say) of pages and an area for a page table. We shall refer to the page spaces in the buffer as _buffer pages_ and shall use b ($0 < b < m+1$) to indicate the b-th buffer page. At any time during a program execution, buffer page b will normally be holding some _active_ resident page r (say) form virtual memory. The b-th entry in the page table is used to hold some descriptive information pertaining to resident page r. We discuss that information next.

It is clear that whether or not a certain page p in virtual memory is resident has to be ascertained from the page table. One could look at each entry in the page table until either a reference to p is found or the table is exhausted. However, if we assign a _class_ number to each page given by

$$\text{class}(p) = p \bmod m + 1,$$

then we can reduce the search effort considerably. For each page class c (say), we keep a list of the resident class c pages. Then the only page table entries that need to be searched are those referring to class c pages. Thus, this device reduces the search overhead costs.

When it comes to swapping out the LRU (Least Recently Used) page, the page table has to indicate which buffer page is holding it. If one maintains a list that indicates the order in which the active pages have been referenced, then the LRU page is simply the one at the end of the list. However, the effort required to maintain such a list seems unwarranted and so we simply record the current time (virtual memory reference number) in the page table entry for each buffer page reference. The LRU page must then be determined by a search.

For convenience, let
  b = The buffer page number,
  r = The number of the resident page held in b,
  c = The class number of r and
  m = The page capacity of the page buffer.
Then we require the following information for entry b in the page table, where $0 < b < m+1$:

1. The virtual address of the resident page r

2. The location of the copy of r in mass storage (if it exists)

3. Flag indicating if the active copy of the resident page is the same as the copy in mass storage (if one exists)

4. The most recent time the page in this buffer position was referenced

5. Number of the buffer page holding the next "younger" class c resident page (if any)

6. Number of the buffer page holding the next "older" class c resident page (if any)

7. The youngest class b resident page (if any are resident)

For flexibility, we permit several VM files to be used simultaneously with a common page buffer. Thus, a file number is combined with the virtual address of the page (item 1) in order to resolve the ambiguity. It could have been incorporated with the mass storage location (item 2); however, the former choice permits page table searching with just one compare item (a generalized virtual address) rather than two (virtual address and file number). For economy of implementation, items 2 and 3 are combined by using the sign for the flag item.

4.2 DYNAMIC PAGE MODIFICATION

Applications frequently can be partitioned into logical phases, some of which require extensive I/O operations and others of which are compute bound with little or no I/O. Consequently, it makes sense to permit the dynamic variation of the main memory space used for virtual memory buffering, i.e., the resident set size, accordingly. A reasonable approach to this is to lump both the page table and the page buffer into the same FORTRAN array and place it in "COMMON". Putting it in blank common is convenient when the space is to be used in a compute bound phase, but jeopardizes the sanctity of the page buffer with respect to inadvertent over-write.

It appears that some users are happy to take the risk of damaging the page buffer in exchange for the convenience of having it in blank common whereas others would prefer to not be troubled with that concern. Consequently, two versions of the system are currently maintained, differing only in the placement of the buffer area. The version having the page buffer in blank common is called the B version and the other, called the L version, holds the buffer table in labelled common /VMBUFT/.

5. I/O SYSTEM

The I/O system used by VMSYST is a set of FORTRAN callable subroutines developed by Felippa [2] at Lockheed's Palo Alto Research Laboratory. This set of subroutines, collectively called DMGASP, interacts directly with the operating system of the host computer in order to carry out a variety of file manipulation operations. Versions of it have been completed for Univac 1100 series computers, CDC CYBER computers (NOS and SCOPE operating systems) and the DEC VAX 11/780 computer (VMS operating system). Early versions of the code had to be written in assembly language but a recent version in FORTRAN 77 has been successfully developed.

The operations that DMGASP provides in support of VMSYST are briefly described in this section. For a complete and up to date discussion, the reader must consult [2]. The presentation here will barely suffice for the construction of a simplified version of DMGASP in order to permit use VMSYST without having DMGASP.

5.1 BASIC OPERATIONS

The format of the basic I/O (data management) operations is as follows:

```
DMxAST(L,M,N)
```

where x can be any of the letters DFPWRHEL. These are interpreted as follows:

D	- Declare (assign, attach, activate, open) a file
F	- Free (release, deactivate, etc.) a file
P	- Position a file to a sector (PRU) location
W	- Write on a file starting at its current position
R	- Read from a file starting at its current position
H	- Establish error handling procedure
E	- Write an end-of-file mark (pertains to tape file)
L	- List internal operational information

The arguments serve varying purposes depending upon which operation is involved. For most of the operations, L is the file index called the <u>logical device index</u> (LDI). The equivalence of this index to a named file is established when the file is declared (via DMDAST). For the read and write operations, the second and third arguments M and N are the data array and its size. Thus, basic I/O operations can be illustrated as follows:

```
CALL DMDAST(FILE, NAME, ORGN)  ; Declare a file
```

```
    CALL DMPAST(FILE, 0, 0)        ; Position it to beginning
    CALL DMWAST(FILE, ARRAY, SIZE) ; Write on it
            ..
    CALL DMPAST(FILE, LOC, 0)
    CALL DMRAST(FILE, ARRAY, SIZE) ; Read from it
            ..
    CALL DMFAST(FILE, 0, 0)        ; Free the file
```

The last two arguments M and N in the position operation DMPAST give the location (in sectors or PRU'S) and mode. For our purposes, the mode N should always be 0. Other values for the mode N allow relative addressing and the use of other addressing units such as words. Similarly, the last two arguments of operation DMFAST should always be 0 for our applications. The third argument is a dummy argument and the second affects special termination conditions such as a final file erasure. Setting the second argument M to zero causes release of the file in a manner appropriate for the way in which it was declared.

DMDAST

File declaration is the most unpleasant I/O operation but, of course, must always be done prior to any other I/O operations on a given file. The form of the operation

```
    DMDAST(FILE, NAME, ORGN)
```

given in the illustration above is used to describe it. The arguments are interpreted as follows:

FILE	- The (integer) index to be used for the file
NAME	- The external device name associated with the file (up to 18 characters <u>left</u>-<u>adjusted</u> and <u>blank</u> <u>filled</u>). The last character must be a blank. If the first character is a blank, a default name is generated.
ORGN	- An integer array describing the organizational aspects of the file as follows:

1 Type of hardware
- = 0, Default, sector addressable, random access device
- = 1, High speed, low capacity (scratch) device
- = 2, Like = 1 case but word addressable device
- = 3, Extended RAM (core) storage
- =-1, Seven track tape (system default density)
- =-2, Nine track tape "

2 Permanency and accessibility attributes

```
                = 0, Temporaty file
                = 1, Write enabled mag. tape
                = 2, Previously assigned file
                = 3, Cataloged file, read only
                = 4, Cataloged file, read or write permitted
                = 5, Create new reserved tape file
                = 6, Catalog new public disc file
                = 7, Catalog new read-only private disc file
                = 8, Catalog new read-only public disc file
                = 9,10,11 or 12, Same as = 5, ..., 8 cases
                     but catalog file only if correct program
                     termination
                =-1,-2, ..., -12, Same as positive cases but
                     first check if the file is already as-
                     signed.
                     If it is, simply establish linkage with
                     the file index.

            3  Device capacity in words (if appropriate).
                 = 0, Use default value (usually best choice)

            4  PRU (physical record unit) size in words. If
               ORGN(4)=0, the physical sector size of the
               storage device will be used. The PRU size
               must be a multiple of the sector size.
               This item is ignored for mag tape.
```

The NAME and ORGN information is obtained directly from the NAME and PARAMS arguments of VMOPEN as described in Section 3.3.2.

5.2 SUPPLEMENTARY OPERATIONS

In addition to the basic I/O operations there are a number of convenient supplementary operations primarily used for information purposes. The supplementary operations used by VMSYST are all integer functions of the forms:

```
    LMLDIU(NO)    - The NOth entry in the list of logical
                    device indexes recognized by DMGASP.
                    (Returns zero if NO is out of range)
    LMUNIT(FILE)  - FORTRAN unit number corresponding to
                    logical device index FILE (if one exists).
                    This is the inverse of LMLDIU, i.e.,
                      FILE = LMLDIU(LMUNIT(FILE)).
                    (Returns zero if FILE is out of range)
    LMLIMT(FILE)  - The capacity of the file in PRU'`S
    LMNEXT(FILE)  - The next free PRU on the file
    LMSECT(FILE)  - The sector size of the file in words
    LMPRUS(FILE)  - The PRU size of the file in words
```

where NO is any integer less than 60 and FILE is a logical device index (integer) linked to a declared file via the DMDAST operation. The correspondence between device indexes, such as NO, and logical device indexes, such as FILE, is internally established by DMGASP. It may vary among installations, even for the same host computer. Logical device indexes are used to avoid problems associated with special (non-standard) purposes given to certain I/O unit numbers for FORTRAN programs such as 5 for the card image reader and 6 for the line printer.

ACKNOWLEDGEMENTS

This project has evolved over the last four years through several versions. The author is indebted to many colleagues that have used and commented on earlier versions of this system. Particular thanks go to Dr. J. K. Reid for originally suggesting this system and for numerous comments on it. Thanks also go to Mr. L. E. Stearns for some of the implementation and to Dr. C. A. Felippa for carefully reading this manuscript and suggesting numerous improvements. Work on this project has been supported by the Air Force Office of Scientific Research, contract F49620-76-C-0003, the Air Force Flight Dynamics Laboratory, contract F33615-76-C-3105, and L'Office National d'Etudes et des Recherches Aerospatiales, contract 19-288-SAT-ONERA.

REFERENCES

1. Denning, P.J., "Virtual Memory," ACM Computing Surveys, 2,3 (1970) 153-189

2. Felippa, C.A., "The Input-output Manager DMGASP," LMSC-D766628, Lockheed Palo Alto Research Laboratory, Palo Alto, (October 1980)

3. Fine, G.H., "Dynamic Program Behavior Under Paging," Proc. 21st ACM Nat. Conf., Thompson Book Co., Washington, D.C. (1966)

4. Goldberg, R.P., "Virtual Machine Systems," Report MS-2687, MIT Lincoln Lab. (Sept. 1969)

5. Hatfield, D.J. and J. Gerald, "Program Restructuring for Virtual Memory," IBM Systs. Jnl. 10 (1971) 168

6. McGrath, M., "Virtual Machine Computing in an Engineering Environment," IBM Systs. Jnl., 11 (1972) 131

7. McKellar, A.C. and E.G. Coffman, Jr.,"Organizing Matrices and Matrix Operations for Paged Memory Systems," Comm. ACM, 12,3 (1969) 153-165

8. Parmelee, R.,"Virtual Machines: Some Unexpected Applications," Proc. 1971 IEEE Int. Comp. Soc. Conf., Boston, Mass. (1971)

9. Parmelee,R., T.I. Peterson C.C. Tillman and D.J. Hatfield, "Virtual Storage and Virtual Machine Concepts," IBM Systs. Jnl., 11 (1972) 99

GLOSSARY OF TERMS

COLD START - Program start with an initially empty resident set

CONTROL PARAMETER - A parameter that governs the size of the resident set and the duration of each segment's (page's) sojourn

LIFETIME CURVE - The mean number of references between page faults versus the resident set size

LOCALITY - The concept that a program favors a subset of its segments during extended time intervals (phases)

LOCALITY SET - The collection of segments used by a program during a phase (see locality)

PAGE - A fixed size block of contiguous locations in a virtual address space

PAGE ALIGNED SEGMENT - A segment that starts at the beginning of a page and ends at the end of a page in virtual address space.

PAGE BUFFER - Region in main memory used to hold the resident set

PAGE FAULT - Interruption of a program to modify its resident set in order to satisfy a reference r(t) at some time t

PAGE TABLE - A resident table used to implement the paging policy

PAGING POLICY - The method used to decide what modification of the reference set will be made in order to service a page fault. Typical policies for page removal are:
FIFO - First in, first out
LRU - Least recently used
ALD - Atlas loop detection

PHYSICAL RECORD UNIT (PRU) - The addressable unit of data used for I/O operations to an auxilliary storage device. The size of a PRU (in words) must be a multiple of the granularity (e.g., sector size) of the storage device.

REFERENCE - A single access (get or put) to a segment

REFERENCE STRING - r(1), r(2),...,r(t) A string (or sequence) of references where, r(t) is the reference at time t

RESIDENT SET - All segments in main memory at a given time. The size of each segment in the resident set is a multiple of the page size.

SAMPLE INTERVAL - A fixed length time interval serving as a unit for measuring memory management performance

SECTOR - The smallest addressable unit of data for an auxilliary storage device such as drum or disc. (Hardware and O/S dependent)

SEGMENT - A block of contiguous locations in virtual address space

SLOW-DRIFT LOCALITY - The proposition that (locality) phases tend to be long and that changes in the locality sets between phases tend to be mild "The trouble [with this concept] is that it is wrong" P.Denning 1978

SWAPPING CURVE - The rate of segment (page) faults versus the resident set size

TIME - A measure of memory references

VIRTUAL ADDRESS SPACE - A sequence (starting with 1) of storage locations for holding words of data that are accessed by means of a virtual storage management system.

VIRTUAL MEMORY - An organized collection of data in computer storage that is accessed by means of a specific hardware and/or software system using elements of a virtual address space.

WARM START - Program start with a non-empty resident set.

WORD - The data unit used to hold a single-precision floating-point number in main storage. Typical word sizes are: 32 bits (DEC and IBM), 36 bits (Univac) and 60 bits (CDC).

WORKING SET - The set of segments (or pages) used during a given sample interval

WORKING SET POLICY - A memory management policy for which the resident sets are always working sets

A COMPARISON OF TWO SPARSE MATRIX PROCESSING TECHNIQUES FOR STRUCTURAL ANALYSIS APPLICATIONS

William A. Loden

ABSTRACT

The performance of algorithms based on two techniques for the solution of large sparse, symmetric linear equation systems of the type that arise frequently in structural analysis are studied. Representative test problems, for simple and complex structural configurations treated with the finite element method, are solved with two implementations of Cholesky method profile algorithms and with the SPSYST sparse-matrix system package developed by J. K. Reid.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

This study compares two techniques for the solution of large sparse, symmetric linear equation systems that arise frequently in structural analysis. The equation system of interest,

$$Ax = b, \tag{1.1}$$

has a symmetric, positive-definite matrix A that can be expressed as

$$A = \sum E^{(k)} \tag{1.2}$$

where the $E^{(k)}$ are very sparse "element" matrices. A matrix is said to be sparse when a large number of its entries are zero.

Direct methods for solution of this basic problem have a long and interesting history, and excellent surveys of work on this subject are available [e.g., 2,3,4]. Profile methods (also called skyline or variable-bandwidth methods) comprise a class of direct methods that has gained widespread usage due to their relative simplicity, flexibility, reliability and reasonable efficiency. Techniques that take fuller advantage of the sparseness of A have been developed comparatively recently and are considerably less well-established. Potentially, they have an unquestioned advantage for very large problems because they involve fewer operations and deal with smaller volumes of data. Their main weakness, to date, has been their complexity. Sparse matrix programs and program systems tend to be collections of complex routines that require considerable skill on the part of the analyst for effective application. As noted by P. S. Jensen in [5], although a number of sparse matrix packages have been constructed in recent years, none appear to be suitable for very large problems that cannot be held within the high speed memory of a computer.

The SPSYST package [1], based on a generalization of the frontal method [6], appears to be one of the most promising algorithms to emerge in recent years. Its internal data management system is based on a virtual memory approach, using a special FORTRAN language virtual storage "simulator" [7] for non-virtual computer systems.

The objective of this study is to compare a generalized wave front technique (SPSYST) with currently popular profile methods for applications to structural analysis. Section 2 of this report begins with brief descriptions of two profile programs used in this study. The programs used were selected because of their ready-availability and because of their wide usage in various large-scale programs and program systems at the Lockheed Missiles and Space Co., Inc. (LMSC). This Section continues with a short description of the profile-minimization algorithm used to produce near-optimum orderings of the equations of A, for use with the profile method approach. Following a brief description of SPSYST, Section 2 concludes with a description of the testing program (called SPARTA) that was constructed to apply SPSYST to problems for which the element matrices are generated by an external source (structural analyzer).

In Section 3, five test problems are described and the performances of SPSYST (i.e., of the SPARTA program in which SPSYST is imbedded) and of the two profile method programs are compared. These problems, the finite element models for which were developed in recent years by various engineering groups at LMSC, range from relatively simple to moderately complex, and are representative of the types of structural analysis problems that occur frequently in practice.

Section 4 includes observations and comments based on experiences gained in this investigation, and offers some suggestions for further work.

SECTION 2

THE PROFILE AND SPSYST ALGORITHMS

2.1 The Profile Algorithms

Profile methods, which evolved from the constant-band matrix techniques developed in the late 1950's and early 1960's, enjoy wide usage with medium- and large-scale finite element and energy-based finite difference structural analysis programs and program systems. For this approach, it is anticipated (but not essential) that the system matrices (i.e., the stiffness, mass, damping and/or other matrices that characterize the system to be treated) have their nonzero entries clustered about the main diagonal. These methods are organized to take advantage of this structure by physically storing and performing arithmetic operations only within the active area of the A matrix [i.e., within that portion of the matrix that lies between the main diagonal and the farthest nonzero off-diagonal entry in each column of the upper triangular half (or, equivalently, in each row of the lower triangular half) of the matrix]. The shape of the active area remains invariant during factorization operations. The popularity of profile methods derives, in part, from the facts that: 1) relatively straightforward management of main and auxiliary storage is required, 2) the storage and run-time costs are easily predicted in advance, 3) they are reasonably efficient for many applications and 4) they can be implemented by relatively short and simple codes.

Commonly used profile algorithms are based on Gaussian elimination, standard Cholesky (A= LL') and modified Cholesky (A= LDL') methods. The implementations differ primarily in the manner in which the nonzero coefficients of A are arranged within the various storage devices (i.e., within core, extended core, on random-access mass storage devices, etc.). These differences become quite significant as the size of A and the storage requirements for the active area of A increase, and the efficiency of most of these algorithms is closely-tied with the architectural characteristics of the computer system being used. Two representative profile algorithms are used for this study, partly to limit the scope of this investigation and partly because of their ready-availability and confidence in results obtained with them at LMSC. The first of these, called "OLDPRO" for convenience, has been used extensively within LMSC's DIAL [8], NEPSAP [9] and REXBAT [10] program systems and is a well-tested, fairly-efficient Cholesky method. The second, called "SKYPUL", is part of a recently-developed set of utility routines [11] for treating large sparse, symmetric equation systems, using the LDL' Cholesky approach.

2.2 Profile-Minimization Algorithm

The efficiency of profile algorithms is closely-related to the topological characteristics of the equation system(s) being treated. Well-ordered systems, for which the active areas are small, can be factored and solved with less effort and at less expense than poorly-ordered systems. It is difficult and uninteresting for the analyst to choose the order in which the equations should be treated in order to minimize the size of the active area. Consequently, one of several effective, efficient and fairly easy-to-use profile minimization algorithms is generally used as a matrix preprocessor. In this study, a profile minimizer called PROM was used to generate good orderings for the profile solutions of the test problems considered. PROM is an implementation of the Gibbs-Poole-Stockmeyer algorithm [12].

2.3 The SPSYST Algorithm

SPSYST is based on a generalization of the frontal method [6], in which the equation ordering for a finite element matrix A (say) for Gaussian elimination is determined by the order in which the elements that generate A are assembled. The factoring of A is carried out simultaneously with its assembly (i.e., with the summation of the elements generating A). The factorization for each row is completed as soon as it is fully assembled. The fundamental assemble/factor process is a pairwise operation in which two matrices are summed and the "internal" variables (i.e., those that then correspond to fully-assembled rows of A) are eliminated before another pair of matrices is considered. The order in which the element matrices are combined thus determines the order in which the rows are eliminated. For a more detailed description, see [1] or [5].

The two versions of SPSYST used in this investigation construct good orderings with minimal information provided by the analyst. One version of SPSYST employs a "minimum fill" heuristic, and the other uses the "minimum degree" approach.

For analyses of configurations in which there are natural groupings of elements into substructures (some of which might be repeated several times in the full structure), SPSYST is designed to accept this information from the user in order to simplify and perhaps reduce the cost of its analysis. This information is presented to SPSYST in a natural way that is convenient for the user.

2.4 The Test Program

In order to apply the SPSYST algorithms to the test problems considered in this study, it was necessary to construct a cover program through which the externally-generated data for the problems (i.e., the element stiffness matrices and information relating them to the displacement variables) could be accessed. Special user controls for specifying element sets to be treated as "user-generated super-elements" (abbreviated UGSEs hereafter) were included. This cover program, called "SPARTA" (for SPARse Testing Algorithm), was designed to accept element-stiffness data from any external source, i.e., any program or program system used to define the configuration to be analyzed, with the only restriction being the fact that these data must be in a certain format and sequentially ordered on an input (mass-storage) file. The REXBAT finite element program system [10] was used to generate all of the structural models considered in this study. These "REXBAT" models, which represent actual structural configurations that have been developed at LMSC in recent years, are representative of the types of structures problems that are likely to be encountered in practice.

The principal features and logical structure of the main program for SPARTA are shown in Figure 2.1. An important feature of this program is its ability to sense the core-space requirements for a problem, and to acquire more space as necessary. This is especially important when SPARTA operates on non-virtual machines such as the UNIVAC 1100/83 system used for this study. This "dynamic sizing" is accomplished by placing all of the problem-size-dependent major arrays within the blank common block, with starting locations at addresses that are determined by the program. These arrays are passed to the SPARTA and to the SPSYST subroutines via calling sequences.

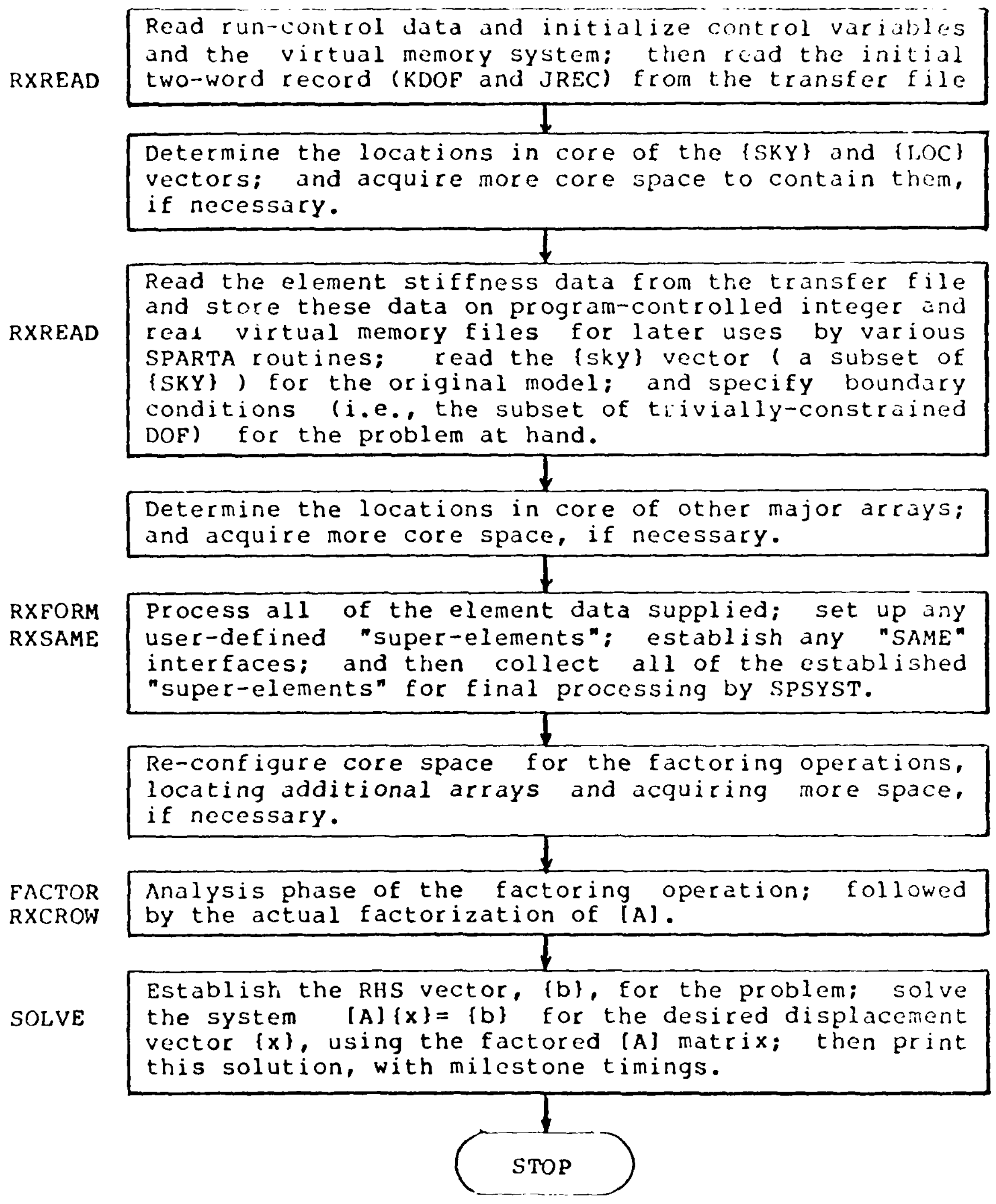

RXREAD
Read run-control data and initialize control variables and the virtual memory system; then read the initial two-word record (KDOF and JREC) from the transfer file
Determine the locations in core of the {SKY} and {LOC} vectors; and acquire more core space to contain them, if necessary.
RXREAD
Read the element stiffness data from the transfer file and store these data on program-controlled integer and real virtual memory files for later uses by various SPARTA routines; read the {sky} vector (a subset of {SKY}) for the original model; and specify boundary conditions (i.e., the subset of trivially-constrained DOF) for the problem at hand.
Determine the locations in core of other major arrays; and acquire more core space, if necessary.
RXFORM
RXSAME
Process all of the element data supplied; set up any user-defined "super-elements"; establish any "SAME" interfaces; and then collect all of the established "super-elements" for final processing by SPSYST.
Re-configure core space for the factoring operations, locating additional arrays and acquiring more space, if necessary.
FACTOR
RXCROW
Analysis phase of the factoring operation; followed by the actual factorization of [A].
SOLVE
Establish the RHS vector, {b}, for the problem; solve the system [A]{x}= {b} for the desired displacement vector {x}, using the factored [A] matrix; then print this solution, with milestone timings.
STOP

Figure 2.1 Schematic of the Main Program for SPARTA.

This replaces the use of various named common blocks (the sizes of which are not adjustable at run time), by the original version of SPSYST. The total size of the blank common block, which (on the UNIVAC computer system) is mapped onto the high end of the program's data bank, is adjusted by means of appropriate executive requests to the computer's operating system. It is thus possible to increase or decrease the total size of the SPARTA program as the core-space requirements increase or decrease during any given analysis.

As noted in the preceding section, the specification of the order in which the elements are to be treated is a critical aspect of this method. This specification is accomplished within a subroutine called "RXFORM". After all of the supplied elements are "processed" (by removing constrained DOF from the lists of variables associated with them and by calling the appropriate SPSYST subroutine (INELV) to get the modified lists of variables into the SPSYST "system") the analyst can do any of the following things:

* Specify one or more UGSE's, which can be "constructed" using elements and/or other UGSEs.

* Generate a "final" UGSE which includes all supplied elements that have not been assigned to other UGSEs (if none of the supplied elements are to be excluded from the analysis at hand).

* Employ SPSYST's "SAME" interfacing capabilities for repeated elements. These operations are controlled by subroutine RXSAME, which facilitates establishing the equivalence of UGSEs to other UGSEs and/or the creation of new "super-elements" the elements of which did not exist in the original model.

After these operations are completed, all of the established super-elements are collected into a final "root super-element".

It is possible for the analyst to specify the entire assembly order (by repeated use of the first process in the above list). This is likely to be a very time-consuming and counter-productive effort, especially for a complex structure. SPSYST is able to construct a good ordering for each super-element that it is called upon to process and, by extension, for a complete problem as it becomes a single root super-element through concatenation of all user- and program-generated super-elements.

SECTION 3

TEST PROBLEMS

3.1 Introduction

In order to compare SPSYST with two representative profile algorithms (OLDPRO and SKYPUL), five test problems were set up and solved in various ways using these solvers. This small set of test problems, four of which use finite element models for structural configurations that were analyzed at LMSC in recent years, is not intended to be comprehensive; but it is expected to represent the types of structural analysis problems that are frequently encountered in practice.

The first problem involves a two-dimensional, triangular domain that models a simple dam structure. This relatively small problem was solved in numerous ways with SPARTA, using different combinations of "user-generated super-elements" (UGSEs) and several uses of SPSYST's "SAME" interfacing technique. The second problem involves a more complex finite element model of a "gyro" cradle assembly and was solved twice with SPARTA, once using a single UGSE and again using three. The third problem involves a finite element model of a structural component used in ocean-mining operations (the "porch" structure) and was only solved once with SPARTA, using a single UGSE. The fourth problem (the "tower" structure) involves a finite element model of a launch umbilical tower and was solved twice, using one and using three UGSEs. The final problem, the "cross-cone" structure (a finite element model of part of a vehicle), was solved twice, using one and using two UGSEs. All of these problems were also solved using the OLDPRO profile algorithm, and two of them were also solved with the SKYPUL profile algorithm.

One measure of the efficiency of any algorithm for matrix factorization is the number of FLOPs (floating point operations, consisting of a single multiplication and addition) required to do the complete process. FLOP counts are given for SPSYST and for the profile algorithms used for each of the problems considered. The number of nonzeros is also given, as a measure of the in-core storage requirements (for real numbers) during the factoring operations with SPSYST, and as a measure of the total storage requirements for the fully-assembled A matrix with the profile algorithms. These measures, by themselves, do not take into account the computational expenses (e.g., the cost per FLOP) for the various algorithms used, and good FLOP counts must be interpreted in the light of that kind of information, too.

In order to measure these computational expenses, calls to a timing routine which measures elapsed central processor (CP) time were made at various key points in SPARTA and in the REXBAT program. Another UNIVAC system timing processor (which measures elapsed CP time, I/O activity and time, and total run cost) was exercised immediately before and after each execution of SPARTA, of REXBAT processors, and of SKYPUL processors. This timing information proved to be repeatable and to correlate well with the actual run costs.

The "dam" problem was solved repeatedly during the program coding and verification efforts as SPARTA was being developed on the DEC VAX-11/780 computer system. Only a few of these tests were repeated when SPARTA was shifted to the UNIVAC 1100/83 computer system for the numerical experiments with the other four problems, so the "dam" results included here do not include "timing" data. This is not critical, since the "dam" problem is the smallest one considered in this study.

3.2 The "Dam" Problem

The finite element model for the "dam" problem (see Figure 3.1) involves 50 identical linear strain (6-node) triangular elements. These elements are connected among 121 node points, each of which has two translational degrees of freedom (DOF), giving a total of 242 DOF in the unconstrained structure. All of the node points along the bottom of the "dam" are fully constrained (i.e., both displacement components are set equal to zero), so the problem has only 200 "active" DOF (unknowns).

The complete structure was analyzed with the REXBAT program, using the OLDPRO profile method for the factorization of the completely assembled stiffness matrix, with no use of that program's substructuring techniques. The relevant results for that analysis are shown in Table 3.1.

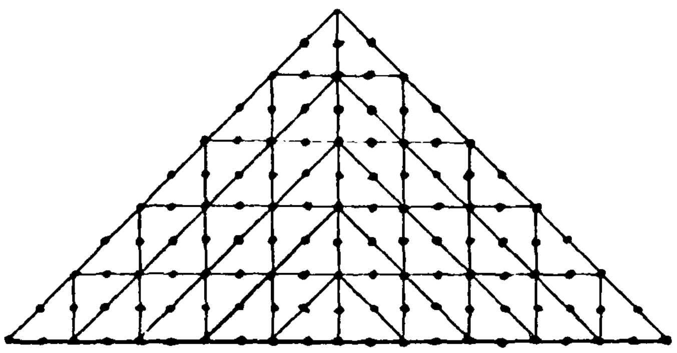


Fig. 3.1 The "Dam" Problem.

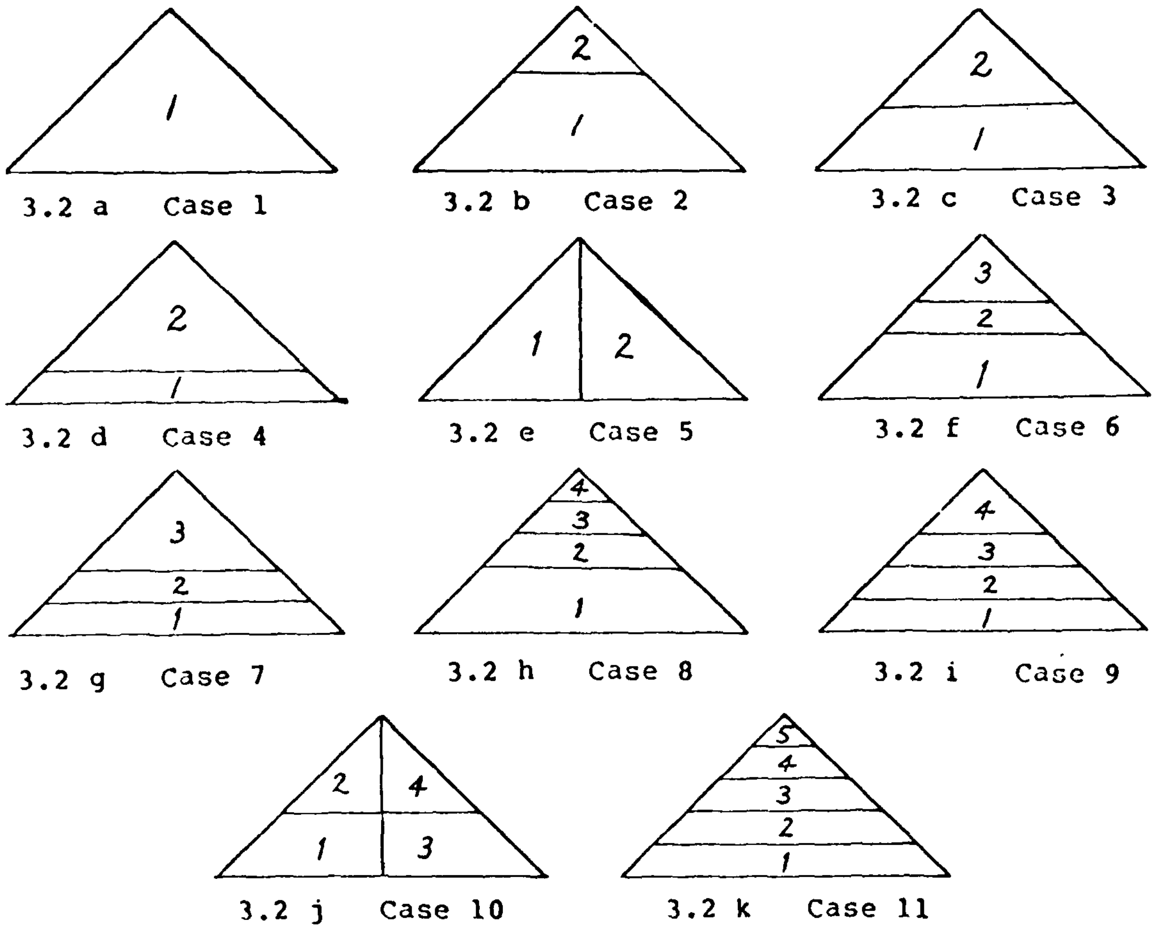
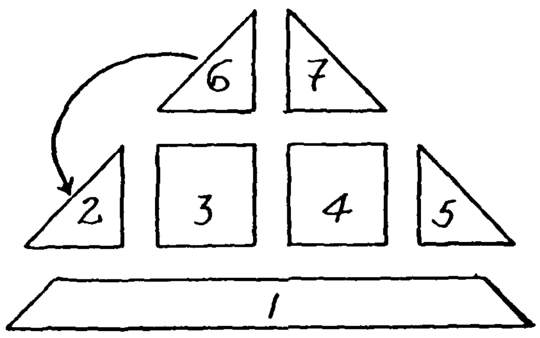
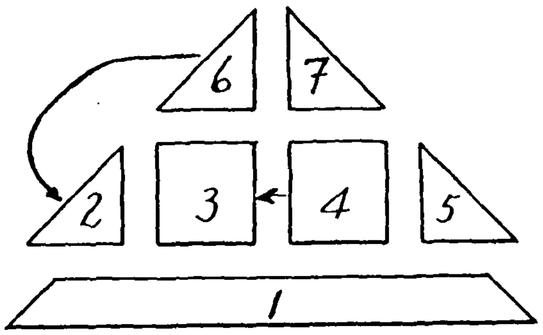


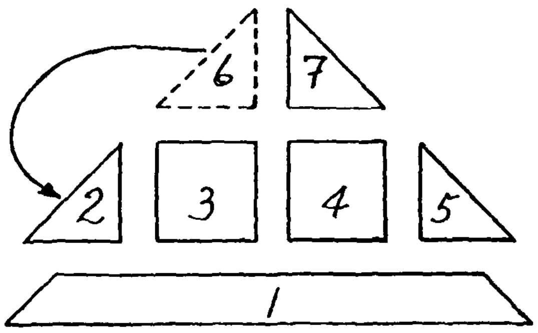
Fig. 3.2 Dam Substructurings for SPARTA Analyses.



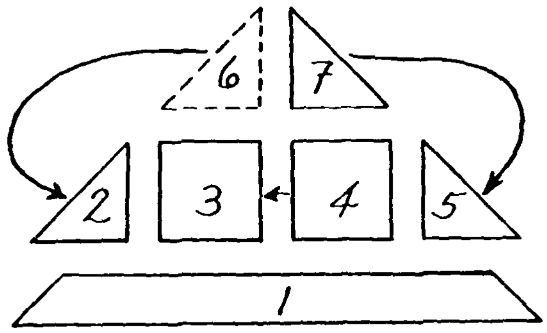
3.3 a   Case 1



3.3 b   Case 2



3.3 c   Case 3



3.3 d   Case 4

Fig. 3.3   SPARTA Analyses of Dam, Using "SAME" Interfaces.

TABLE 3.1

PROFILE SOLUTION OF THE DAM PROBLEM

Total no. of DOF	242
No. of "active" DOF	200
Maximum Semibandwidth of A	44
Average Semibandwidth of A	24.6
Nonzero Stiffness Values	3400
Active Area	5943
Factorization FLOPs	67896

This problem was analyzed with SPARTA in 11 different ways wherein the full structure (50 elements) was treated as a single "user-generated super-element" (UGSE) and as assemblies of two, three, four and five UGSEs, with the elements used to form these UGSEs taken in different combinations. The substructurings considered are shown in Figure 3.2, and the UGSE sizes and the FLOP counts for these analyses are shown in Table 3.2.

TABLE 3.2

FLOP COUNTS FOR SPARTA SOLUTIONS OF THE DAM PROBLEM

Case No.	Figure No.	Number of Elements Included in UGSE No. 1	No. 2	No. 3	No. 4	No. 5	No. of FLOPs
1	3.2 a	50	-	-	-	-	28548
2	3.2 b	42	8	-	-	-	27428
3	3.2 c	32	18	-	-	-	32892
4	3.2 d	18	32	-	-	-	54828
5	3.2 e	25	25	-	-	-	28484
6	3.2 f	32	10	8	-	-	37244
7	3.2 g	18	14	18	-	-	67396
8	3.2 h	32	10	6	2	-	38652
9	3.2 i	18	14	10	8	-	71748
10	3.2 j	16	9	16	9	-	33260
11	3.2 k	18	14	10	6	2	73156

The lowest FLOP counts obtained here (27428 and 28484) were for two cases in which the complete structure was divided into two UGSEs by the analyst, but these counts are not much lower than the result (28548) that was obtained with all of the elements used to form a single UGSE. Most of these results are lower than the FLOP count (67896) obtained for the profile method.

This problem was also analyzed with SPARTA in 4 different ways wherein the full structure was treated as an assembly of 7 UGSEs, with the substructuring scheme shown in Figure 3.3, and using SPSYST's "SAME" interfacing capabilities to equivalence some of these substructures to others. In the first two of these analyses, all 50 of the REXBAT-generated element matrices were used in forming the 6 UGSEs; and one or two of these UGSEs were declared to be the "same" as one or two of the other ones. In the third analysis, the elements comprising UGSE number 6 were assumed not to be given; and the program was required to "create" the missing substructure by using one that is the "same" as UGSE number 2. In the fourth analysis, two UGSEs were taken to be the "same" as two others; and UGSE number 6 was "created" as in the third analysis. The pertinent results for these tests are given in Table 3.3. The FLOP result for this substructuring, without using the "SAME" interfacing capability, is also included in this Table as Case 5.

TABLE 3.3

SPARTA FLOP COUNTS FOR DAM, WITH "SAME" INTERFACING

Case No.	Figure No.	UGSE No.	or PGSE No.	...is equivalenced to UGSE No.	No. of FLOPs
1	3.3 a	6	-	2	66382
2	3.3 b	6	-	2	
		4	-	3	54245
3	3.3 c	-	6	2	63838
4	3.3 d	4	-	3	
		7	-	5	
		-	6	2	59759
5	-----	-	-	-	58612

PGSE= "program-generated super-element"

The FLOP counts for Cases 1 through 4 are not significantly different from that for Case 5, with which they should be compared. All of these cases have higher FLOP counts than the best of the previously-described cases (in which "SAME" interfacing was not attempted). These last tests served primarily to verify the correctness of the implementation, in SPARTA, of the "SAME" interfacing process. The relative "efficiency" and other aspects of this technique must be determined through other tests (which, unfortunately, were not performed in the present study).

3.3 The "Gyro" Problem

The finite element model for the second ("gyro") problem (shown in Figure 3.4) has 323 elements (including 89 12-DOF beams, 104 18-DOF triangles, 126 24-DOF quadrilaterals and 4 12-DOF hand-generated elements). These elements are connected among 179 node points, each of which has six DOF; and the problem has a total of 1074 DOF of which 1038 are "active", since six node points are fully-constrained (i.e., have their displacement values fixed to be zero).

The complete problem was analyzed with the REXBAT program in two ways: once (Case 1) with the DOF ordering supplied by the original analyst (who tried to produce as good an ordering as possible without making extraordinary efforts to do so), and again (Case 2) with the DOF ordering that was determined by the PROM profile-minimization program. The PROM-optimized version of the problem was also analyzed with the SKYPUL processors (Case 3), which involved: 1) conversion of the assembled A matrix from its OLDPRO form to the SKYPUL format, 2) factorization of this matrix, 3) generation of an appropriate RHS vector, and 4) solution of the equation system for the desired displacements. The problem-size and FLOPs results for these profile-method solutions are given in Table 3.4, and the timing results are given in Table 3.5.

TABLE 3.4

SIZES AND FLOPs FOR THE PROFILE SOLUTIONS OF THE GYRO PROBLEM

ITEM	Case 1 OLDPRO	Case 2 OLDPRO + PROM	Case 3 SKYPUL + PROM
Total No. of DOF	1074	1074	1074
No. of "Active" DOF	1038	1038	1038
No. of Nonzero Stiffness Values	44937	44937	44937
Maximum Semibandwidth of A	612	138	138
Average Semibandwidth of A	100.9	82.6	82.6
Profile Area of A	108339	88683	88683
FLOPS for Factorization of A	6211852	3829558	3829558

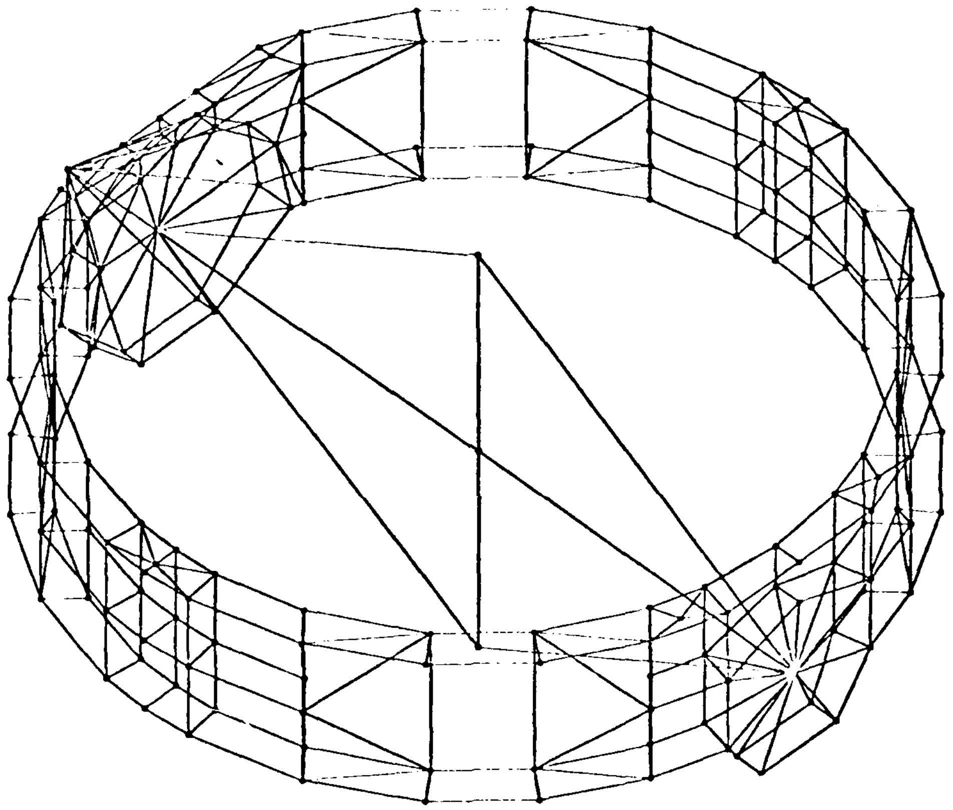


Figure 3.4 Finite Element Model for the "Gyro" Problem.

TABLE 3.5

TIMINGS FOR THE PROFILE SOLUTIONS OF THE GYRO PROBLEM

ITEM	Case 1 OLDPRO	Case 2 OLDPRO + PROM	Case 3 SKYPUL + PROM
Assembly of the A Matrix:			
CP Time [seconds]	4.4	4.2	4.2
I/O Time	39.8	37.6	37.6
SUP Units ($)	48.6	46.9	46.9
Factorization of the A Matrix:			
CP Time	-	-	10.7
I/O Time	-	-	9.5
SUP Units	-	-	25.0
Solving Operations:			
CP Time	-	-	0.6
I/O Time	-	-	6.4
SUP Units	-	-	12.3
Factorization + Solving:			
CP Time	43.8	33.6	11.3
I/O Time	73.0	51.5	15.9
SUP Units	130.1	98.3	37.3
I/O Operations	1113	908	376
Kilowords Transferred	4605	2731	1182
Avg. Cost (SUP/megaFLOP)....	20.9	25.7	9.7

$ A "SUP" is the Standard Unit of Processing on the UNIVAC Computer System in Operation at LMSC at the Present Time

The superior timings obtained with the PROM-generated DOF ordering are evident here; but it should be noted that these timings do not include the extra work involved in generating the required input for the PROM program and the (low) cost of executing PROM to obtain the optimized DOF data for the OLDPRO and SKYPUL programs. The SKYPUL results are significantly better than the OLDPRO results for this problem. The OLDPRO timings are slightly inflated by the fact that the results given were obtained through the UNIVAC system timing processor, which is external to the program module in which the factorization and solving operations are performed, and include other operations (specification and imposition of boundary conditions, computation of nodal reactions, and printing of these data and the displacement solution obtained). Most of the differences, however, must be attributed to SKYPUL's superior logical organization, which (among other things) minimizes the I/O requirements for factorization of A, and to its extensive use of machine-language coding. The OLDPRO routines, like SPARTA, only use machine language in the inner-product operations done primarily within the innermost loops of these programs.

The "gyro" problem was analyzed in SPARTA in two different ways: once (Case 4) where all of the elements of the structure were combined into a single UGSE (letting SPSYST do all of the work in generating the ordering for its "root super-element"), and once (Case 5) with three UGSEs. The substructuring used for this second case, shown in Fig. 3.5, had 12 elements in UGSE number 1 (the cross-member), 163 elements in UGSE number 2 (part of the cradle) and 148 elements in UGSE number 3 (the remainder o' the cradle). The substructuring data, FLOP counts, and timing results for these analyses are given in Table 3.6.

TABLE 3.6

SPARTA RESULTS FOR THE GYRO PROBLEM

ITEM	Case 4	Case 5
Number of Elements in UGSE 1	323	12
Number of Elements in UGSE 2	-	163
Number of Elements in UGSE 3	-	148
No. of FLOPs to Factor the A Matrix	2414455	2564791
No. of Nonzeros for A Factorization	67029	69225
CP Time for "Analysis" [seconds]	3.7	3.9
CP Time for "Factoring"	15.3	15.6
CP Time for "Solving"	1.3	1.2
Total CP Time for SPARTA Analysis	22.2	22.4
Total I/O Time for SPARTA	143.1	145.1
Total SUP Units	176.3	178.6
Total I/O Operations	3467	3672
Kilowords Transferred	1531	1555
Avg. Cost (SUP/megaFLOP)	73.0	68.9

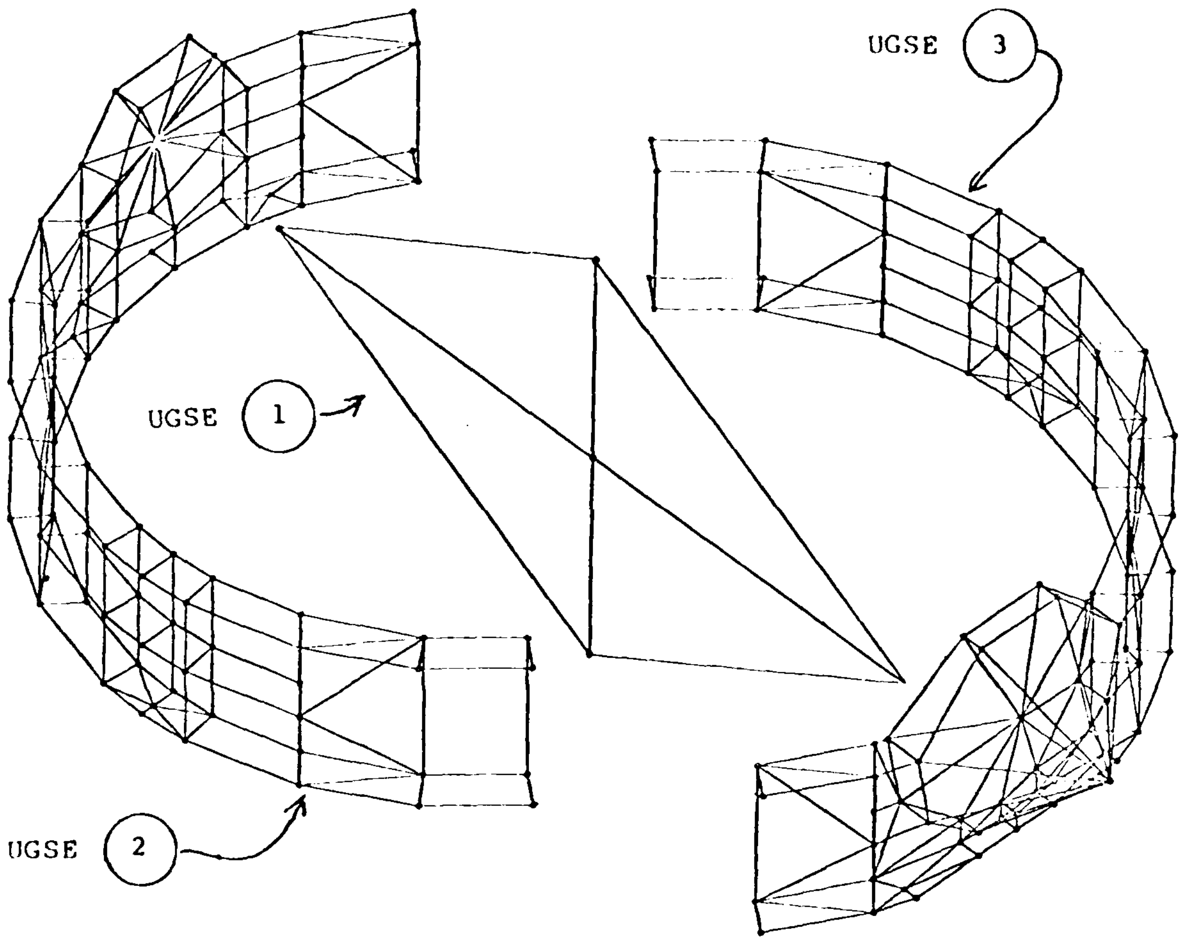


Figure 3.5 SPARTA Substructuring for the "Gyro" Problem (Case 5).

The factoring operation is done in two phases in SPARTA: the first part is an "analysis" phase in which the integer lists for the various "super-elements" are examined and preparations are made for the real-number crunching, and the second phase (which might be done many times, using different sets of real numbers for which the analysis results of phase one apply, in some kinds of problems) where the actual number-crunching is done. The "Total" figures cited here include necessary problem-setup and printing operations, too.

3.4 The "Porch" Problem

The finite element model for the "porch" problem (shown in Fig. 3.6) has 324 elements (including 72 6-DOF axial "bar" elements, 158 12-DOF beams, and 94 18-DOF triangular plates). These elements are connected among 122 node points, most (but not all) of which have six DOF. The problem involves a total of 700 DOF, with 9 of these DOF subjected to "trivial" constraints (i.e., forced to remain zero), leaving 691 "active" DOF for the analysis.

The REXBAT program was used to set up this problem in two ways: the first (Case 1) analysis, which was not carried out to completion, used the original analyst's DOF-ordering scheme (which proved to be singularly uninspired and inefficient); and the second (Case 2) used the DOF-ordering produced by the PROM program. The problem-size and FLOPS results for these two cases, and the timing results for the second (more-reasonable) case, are given in Table 3.7.

The "porch" problem was analyzed once with SPARTA, collecting all of the elements of the configuration into a single UGSE. The FLOP count and timing results from this analysis are given in Table 3.8.

The FLOP-count result for this problem is better in the SPARTA analysis than in the OLDPRO/PROM analysis; but the CP, I/O and SUP timings are all better with OLDPRO/PROM than with SPARTA. This appears to be a problem for which the greater overhead involved in SPSYST's sparse-matrix operations makes SPARTA less efficient than the profile algorithm.

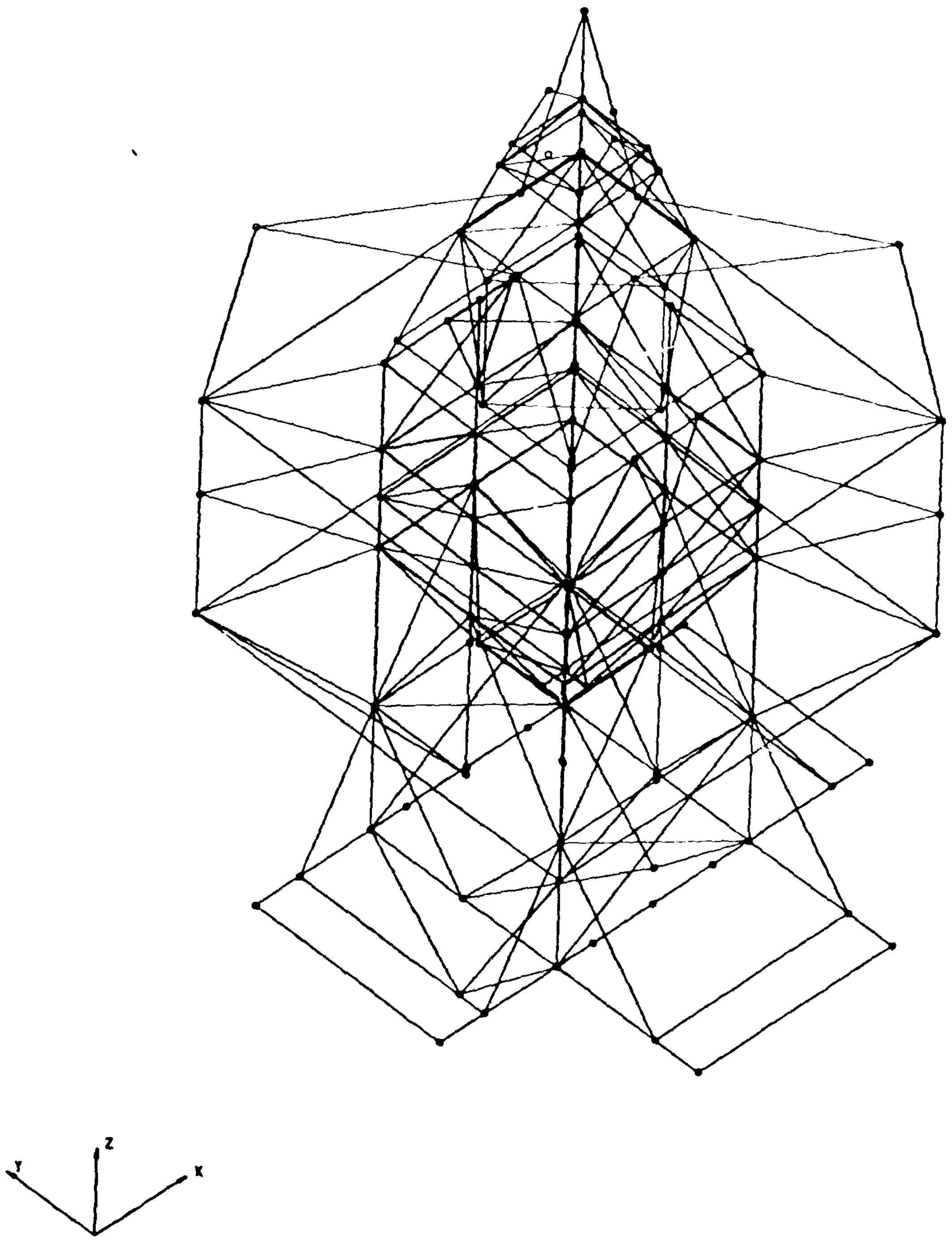


Figure 3.6 The "Porch" Configuration.

TABLE 3.7

SIZES, FLOP COUNTS, AND TIMINGS FOR OLDPRO SOLUTIONS OF THE PORCH PROBLEM

ITEM	Case 1	Case 2
Total No. of DOF	700	700
No. of "Active" DOF	691	691
No. of Nonzero Stiffness Values ...	14499	14499
Maximum Semibandwidth of A	682	138
Average Semibandwidth of A	119.3	64.5
Profile Area of A	83530	45375
FLOPs for Factorization of A	5709522	1714008
Assembly of the A Matrix:		
CP Time [seconds]	-	4.2
SUP Units	-	16.0
Factorization + Solving:		
CP Time	-	13.6
I/O Time	-	6.6
SUP Units	-	33.7
I/O Operations	-	213
Kilowords Transferred	-	1091
Avg. Cost (SUP/megaFLOP)	-	19.7

TABLE 3.8

SPARTA RESULTS FOR THE PORCH PROBLEM

ITEM	Case 3
Number of Elements in UGSE 1	324
No. of FLOPs to Factor the A Matrix	1168776
No. of Nonzeros for A Factorization	35310
CP Time for "Analysis" [seconds]	12.4
CP Time for "Factoring"	11.4
CP Time for "Solving"	0.9
Total CP Time for SPARTA Analysis	28.0
Total I/O Time for SPARTA	24.8
Total SUP Units	61.3
Total I/O Operations	1587
Kilowords Transferred	743
Avg. Cost (SUP/megaFLOP)	52.5

3.5 The "Tower" Problem

The finite element model for the "tower" problem (shown in Fig. 3.7) has 944 12-DOF beam elements connected among 372 node points. Each node has six DOF, giving a total of 2232 equations in the A matrix; but four of these nodes are fully-restrained, so the problem has only 2208 "active" DOF.

This configuration was analyzed with OLDPRO using the DOF ordering supplied by the original analyst (which could not be improved by the PROM program), and with the SKYPUL processors, after converting the A matrix from its OLDPRO format to the SKYPUL format. The problem-size data, FLOP count, and timing results are given in Table 3.9.

TABLE 3.9

SIZES, FLOP COUNT, AND TIMINGS FOR PROFILE SOLUTIONS OF THE TOWER PROBLEM

ITEM	Case 1 OLDPRO	Case 2 SKYPUL
Total No. of DOF	2232	2232
No. of "Active" DOF	2208	2208
No. of Nonzero Stiffness Values	33812	33812
Maximum Semibandwidth of A	150	150
Average Semibandwidth of A	56.5	56.5
Profile Area of A	126108	126108
FLOPs for Factorization of A	3930360	3930360
Assembly of the A Matrix:		
CP Time [seconds]	5.7	5.7
SUP Units	48.0	48.0
Factorization + Solving:		
CP Time	22.9	10.8
I/O Time	52.8	18.9
SUP Units	88.0	40.1
I/O Operations	661	392
Kilowords Transferred	4029	1445
Avg. Cost (SUP/megaFLOP)	22.4	10.2

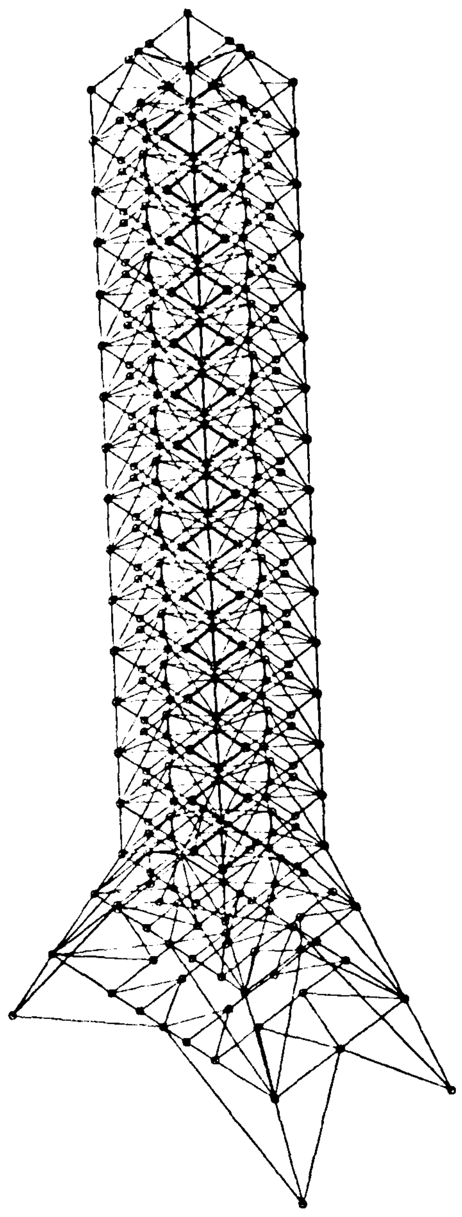


Figure 3.7 The "Tower" Configuration.

The superiority of SKYPUL over OLDPRO is again evident. The important point, however, is that the better profile method requires 5.7 + 10.8 = 16.5 seconds of CP Time and costs 48.0 + 40.1 = 88.1 SUP units to do the assembly and factor/solve operations for this problem.

The "tower" configuration was analyzed twice with SPARTA: first using all 944 elements in a single UGSE (Case 3), and then breaking the complete structure into three UGSEs (Case 4, shown in Fig. 3.8), with 163 elements use in UGSE number 1 (at the base of the tower), 261 elements in UGSE number 2 (just above the base), 260 elements in UGSE number 3, and the remaining 260 elements in UGSE number 4 (at the top of the tower). The results for these two SPARTA analyses are given in Table 3.10.

TABLE 3.10

SPARTA RESULTS FOR THE TOWER PROBLEM

ITEM	Case 3	Case 4
Number of Elements in UGSE 1	944	163
Number of Elements in UGSE 2	-	261
Number of Elements in UGSE 3	-	260
Number of Elements in UGSE 4	-	260
No. of FLOPs to Factor the A Matrix	4593808	4436992
No. of Nonzeros for A Factorization	131784	112590
CP Time for "Analysis" [seconds]	13.9	9.4
CP Time for "Factoring"	26.4	27.3
CP Time for "Solving"	2.1	2.3
Total CP Time for SPARTA Analysis	46.1	42.3
Total I/O Time for SPARTA	296.0	260.0
Total SUP Units	358.2	317.6
Total I/O Operations	7163	6300
Kilowords Transferred	3163	2782
Avg. Cost (SUP/megaFLOP)	78.0	71.6

The FLOP-count results for the profile algorithms and for SPARTA are very close to each other, for this problem; but the CP and SUP timings are significantly lower with SKYPUL (and with OLDPRO) than with SPARTA, so the profile methods appear to be clear "winners" here.

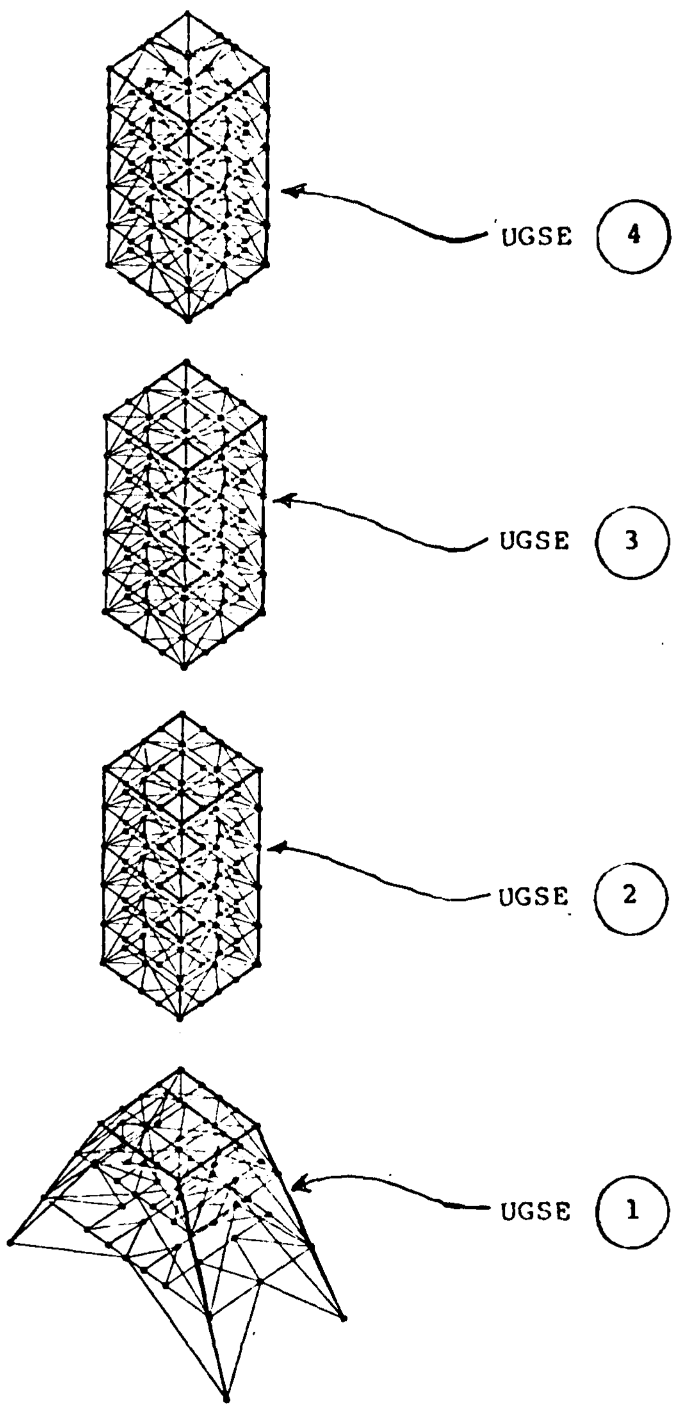


Figure 3.8 SPARTA Substructuring for the "Tower" Problem.

3.6 The "Cross-cone" Problem

The finite element model for the "cross-cone" problem (shown in Figure 3.9) has 864 elements (including 72 6-DOF axial "bar" elements, 10 12-DOF beams, 46 18-DOF triangular plates, and 556 24-DOF quadrilateral plate elements), connected among 719 node points with which a total of 3491 DOF are associated. Most of these node points have six DOF; but some of them have only five, a few have three, and some have none at all. The problem is made somewhat more complex by the fact that some nodes "share" some (but not necessarily all) of their DOF with other nodes, taking advantage of a REXBAT modelling capability that allows the analyst to simulate hinges, ball joints and certain kinds of mechanisms. This DOF-sharing presents no special difficulties to SPARTA. With 68 of the 3491 DOF "removed" by virtue of the applied boundary conditions, there are 3423 "active" DOF in the equation system to be solved.

TABLE 3.11

SIZE, FLOP COUNT, AND TIMING RESULTS
FOR PROFILE SOLUTIONS OF THE CROSS-CONE PROBLEM

ITEM	Case 1 OLDPRO	Case 2 SKYPUL
Total No. of DOF	3491	3491
No. of "Active" DOF	3423	3423
No. of Nonzero Stiffness Values ..	113062	113062
Maximum Semibandwidth of A	958	958
Average Semibandwidth of A	117.0	117.0
Profile Area of A	618039	618039
FLOPs for Factorization of A	57488804	57488804
Assembly of the A Matrix:		
CP Time [seconds]	15.7	15.7
I/O Time	157.1	157.1
SUP Units	180.2	180.2
Factorization of the A Matrix:		
CP Time	-	157.9
I/O Time	-	85.6
SUP Units	-	249.6
Factorization + Solving:		
CP Time	326.0	161.1
I/O Time	288.7	116.6
SUP Units	628.9	290.1
Avg. Cost (SUP/megaFLOP)	10.9	5.1

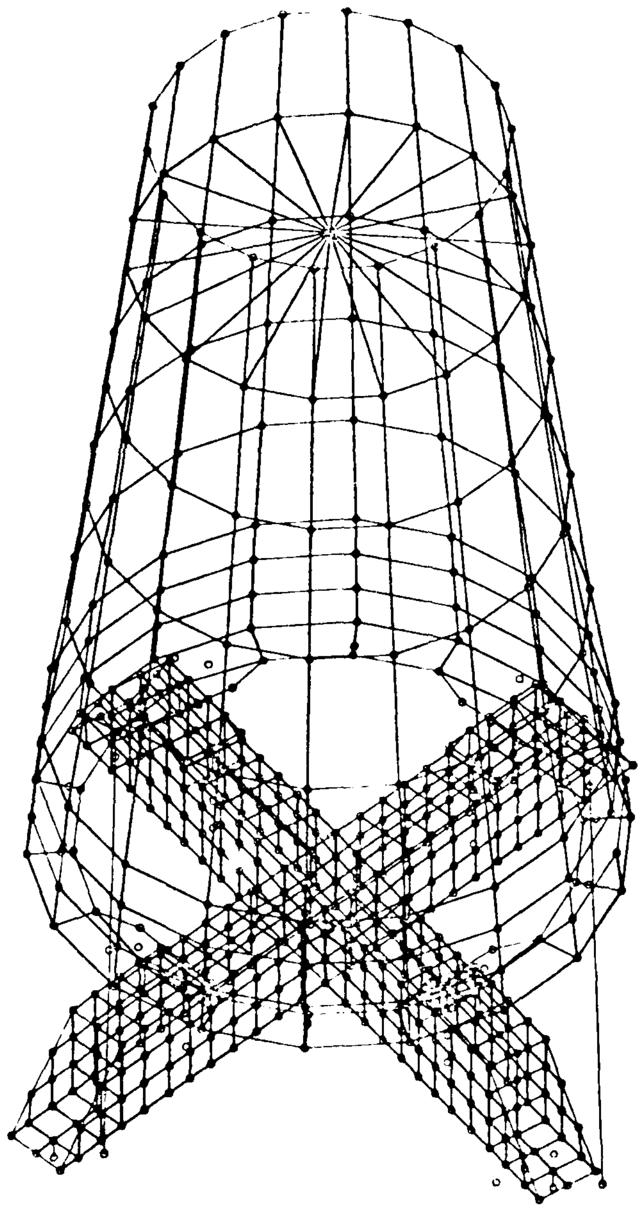


Figure 3.9 Finite Element Model for the "Cross-Cone" Problem.

Here again, SKYPUL's superiority over OLDPRO is evident; but the important results are the total CP, I/O and SUP times for REXBAT assembly plus SKYPUL factorization and solution of the equation system for this problem (176.8, 273.7 and 470.3, respectively).

This configuration was treated by SPARTA in two ways: first using all 684 elements in a single UGSE (Case 3), then breaking the complete structure into two UGSEs (Case 4), as shown in Figure 3.10. In this latter analysis, there were 182 elements in UGSE number 1 (the conical part of the structure), and the remaining 502 elements were in UGSE number 2. The substructuring data, FLOP counts and timing results for these two SPARTA analyses are given in Table 3.12.

TABLE 3.12

SPARTA RESULTS FOR THE CROSS-CONE PROBLEM

ITEM	Case 3	Case 4
Number of elements in UGSE 1	684	182
Number of elements in UGSE 2	-	502
No. of FLOPs to Factor the A Matrix ...	9417382	11110187
No. of Nonzeros for A Factorization ...	229605	248807
CP Time for "Analysis" [seconds]	9.8	8.9
CP Time for "Factoring"	49.9	54.6
CP Time for "Solving"	1.4	4.2
Total CP Time for SPARTA Analysis	65.3	72.1
Total I/O Time for SPARTA	414.9	511.3
Total SUP Units	503.1	604.7
Total I/O Operations	10178	12439
Kilowords Transferred	4367	5412
Avg. Cost (SUP/megaFLOP)	53.4	54.4

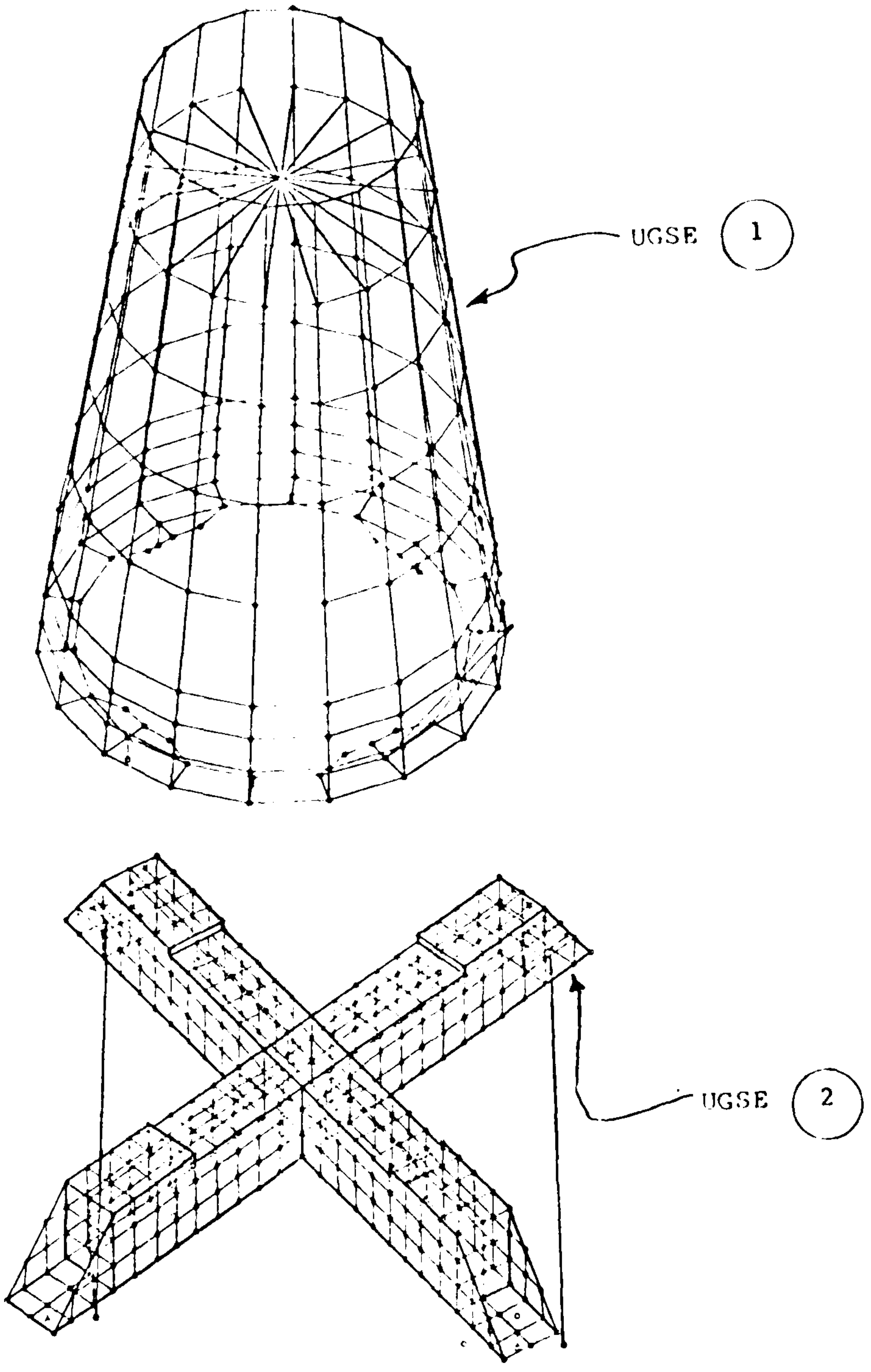


Figure 3.10 SPARTA Substructuring for the "Cross-Cone" Problem.

The FLOP counts and CP timing results for both of these SPARTA analyses are signifigantly better than with either of the profile algorithms, but the I/O and total SUP results with SKYPUL are clearly superior to those with SPARTA.

In the penultimate attempt to run Case 4 (with two UGSEs), a data error caused 15 elements that should have been in UGSE number 1 to be assembled into UGSE number 2 instead; and the SPARTA analysis with this inefficiently-ordered system required more than 27 million FLOPs for the factorization and generated CP and I/O charges of 137 and 1260 seconds, respectively, and 1440 SUP billing units.

3.7 Discussion

All of these analyses were conducted using single precision arithmetic, with double-precision accumulation in the inner-product routines used by OLDPRO and SPARTA (in the machine-language coding). The "porch" and "cross-cone" problems are not very well-conditioned and should have been treated with double precision arithmetic throughout the analysis (on the UNIVAC computer system, which has a 36-bit word size). Double precision results obtained with OLDPRO are available but are not reported here because a double precision version of SPARTA is not available yet.

In all of the OLDPRO and SKYPUL analyses, the largest submatrix block for the partitioned A matrix was limited to be 10000 words or less. Larger blocks would have resulted in less I/O activity during the factorization of A for the larger problems, at the cost of more core space. This is more true for the OLDPRO implementation than for the SKYPUL system, since SKYPUL already uses more submatrix blocks at any given time than OLDPRO does.

All of the analyses conducted with SPARTA were done with a version of the program in which the virtual memory simulator (VMSYST) was an older version of the one described in [7], since the improved version described there was not available yet. In SPARTA, the VMSYST system had an 8960-word dedicated "buffer" block, divided into 20 448-word pages [cf. 7]. A significantly larger buffer space, with more 448-word pages or an appropriate number of larger pages, would substantially decrease the amount of I/O activity required in SPARTA for the larger problems. Here, too, there would be additional costs associated with the use of more core space to accomplish this.

Considering the question of core space, it is instructive to note the core-space requirements for the three programs with which these analyses were performed. The total sizes (including the instructions and program-controlled data space) of OLDPRO, SKYPUL and SPARTA for the four largest test problems are given in Table 3.13. Here, it is apparent that SPARTA tends to require much more core space than either of the profile methods for the largest of these problems.

All of the SPARTA results reported herein were obtained with the version of the program in which the minimum fill heuristic is used in constructing the binary tree for ordering the elimination of elements. Results obtained with the minimum degree version of the program are not sufficiently-reliable to be reported at this time.

TABLE 3.13
PROGRAM SIZES FOR THE LARGEST TEST PROBLEMS

Test Problem	OLDPRO	SKYPUL	SPARTA
Gyro	44420	54271	44026
Porch	43903	-	44026
Tower	43754	55295	81808
Cross-Cone	45341	57343	71056

Program sizes are given in 36-bit words

SECTION 4

CONCLUSIONS AND RECOMMENDATIONS

The primary purpose of this investigation was to compare the SPSYST generalized-wave-front sparse matrix algorithm with the best available profile algorithms for treating large sparse, symmetric linear equation systems of the type that are generated in structural analysis problems. To accomplish this, a convenient testing program (SPARTA) was developed; and a number of test problems were analyzed with SPARTA (which uses the SPSYST sparse matrix package) and with two profile algorithms, OLDPRO (in REXBAT) and SKYPUL.

The profile-method solutions, as might be expected, were generally less-costly than those obtained with SPARTA for the smaller problems; but for the most complex test problem (i.e., the "cross-cone" configuration), SPARTA was significantly faster than both profile method programs (looking at CP time) and just slightly more expensive (looking at the total processing costs) because of the higher I/O requirements required for SPSYST operations. These results are very encouraging, especially in the light of the fact that SPARTA's I/O requirements would be substantially reduced with a larger virtual-memory buffer space than was used in these tests.

Previous work with SPSYST [5] indicated that the program's performance might be noticeably improved through the introduction of improvements in several areas. That work is being carried out by Dr. John K. Reid at AERE Harwell, England, and unfortunately was not completed in time for this study; so the tests reported here were carried out with the older version [1] of the program.

The SKYPUL profile routines proved to be highly-efficient for the problems considered in this study. Some of this efficiency results from the use of the PROM program as a pre-processor in preparing the finite element models for construction of the REXBAT and SKYPUL formats of the assembled A matrices for these problems. Much of it, though, must be attributed to the sophisticated use in SKYPUL of extensive machine-language coding, an advantage that was not shared by OLDPRO and SPARTA.

Notwithstanding this factor, it should be noted that the profile programs proved to be much easier to use (from the analyst's standpoint) than the SPARTA program, for some of the problems treated here. The well-oiled machinery of these extensively-used analysis tools permitted the analyst to use them without a lot of head-scratching and preparation of additional (super-element definition) data. With SPARTA, on the other hand, several iterations were necessary (for the larger problems) in order to ensure that the problem-dependent arrays (the a priori size predictions of which were difficult, at best) could be accommodated within the available core space. Substructuring attempts that "looked good on paper" often proved to be considerably less effective than expected, with the best results generally obtained through the use of only one "user-generated super-element" (i.e., by letting SPSYST do all of the work involved in determining the order in which the elements comprising A were to be treated). SPSYST also proved to be somewhat "sensitive" to errors in the preparation of substructuring data, giving substantially higher FLOP counts and execution costs in two of these test problems when only one element (in one case) or a small group of elements (in the other case) were inadvertently assigned to the wrong substructures.

From various experiences gained in this study, the following additional observations and recommendations are appropriate:

1. The SPARTA program, which accepts externally-generated element data (i.e., stiffness matrices and their associated variable lists) presented to it in a convenient form, can be used in conjunction with a variety of general-purpose structural analysis programs and will be of considerable use in future research studies and for production applications of the SPSYST algorithm.

2. The improved SPSYST routines under development at AERE Harwell should be substituted for the present ones in SPARTA as soon as they are available.

3. A double precision arithmetic version of the improved SPSYST should be implemented in SPARTA at the earliest opportunity.

4. The set of test problems considered in this study should be expanded to include some that are more complex than the present ones, and to include some with which the "SAME" interfacing capabilities of SPSYST can be exercised more meaningfully than in the present study.

5. Tests should be conducted with the improved VMSYST virtual memory simulation program, and using larger VMSYST buffer spaces and more pages than were used in the present study.

6. Programs, like SPARTA, that use SPSYST would benefit greatly from the availability (as output from an SPSYST pre-processing routine) of accurate core-space requirement data. This would facilitate the more efficient utilization by the host program of expensive core-space resources. Analysts would also appreciate the benefits of more meaningful diagnostic messages from SPSYST.

7. Some SPSYST operations would be performed significantly more efficiently with selective use of assembly-language coding instead of FORTRAN in a few key locations in the program. This violates the spirit and intent of code-portability, of course; but if these "key locations" can be effectively isolated and insulated, there should be no serious objections to the use of machine-dependent coding in these few places (in production versions of the program but not necessarily in the research and development versions).

8. SPSYST's performance, especially in the most complex problem considered here, was encouragingly good. With the additional gains anticipated from the program improvements being introduced at AERE Harwell and with the use of larger buffer spaces in the improved version of VMSYST, it is not unreasonable to expect that SPSYST can be the nucleus of an effective and robust tool for structural analysis applications. It looks especially promising for use with problems in which repeated "number crunchings" are performed with topologically-invarient equation systems (e.g., in the extraction of many eigensolutions for complex structural configurations).

ACKNOWLEDGEMENTS

The author wishes to thank Dr. J. K. Reid for his helpful suggestions and comments at the outset of this investigation, and Dr. P. S. Jensen for his guidance and help throughout it and for his careful reading and many suggestions for the improvement of this manuscript. Work on this project was supported by the Air Force Flight Dynamics Laboratory, contract F33615-76-C-3105.

REFERENCES

1 Reid, John K., "A Package of Subroutines for Solution of Very Large Sets of Linear Finite-Element Equations", report AERE-M2947, Atomic Energy Research Establishment, Harwell, England (Feb. 1978)

2 Willoughby, R.A., Survey of sparse matrix technology, in Tech. Report AFFDL-TR-71-9, Air Force Flight Dynamics Lab., Wright-Patterson AFB, Ohio, 65-171 (June 1971)

3 Reid, J.K. (ed.), "Large Sparse Sets of Linear Equations", pp. 17ff, 41ff, 97ff and 255ff, Academic Press, London (1971)

4 George, A., "A Survey of Sparse Matrix Methods in the Direct Solution of Linear Equations", Proc. 1973 Summer Computer Simulation Conf., Montreal, Canada, 15-20 (July 1973)

5 Jensen, Paul S. and John K. Reid, "Sparse Symmetric Matrix Processing", LMSC Report D626184 (May 1978)

6 Irons, B.M., "A Frontal Solution Program for Finite Element Analysis", Int. J. Num. Meth. in Engrg., v2, pp. 5-30 (January 1970)

7 Jensen, P.S., "A FORTRAN Virtual Storage Simulator for Non-Virtual Computers", LMSC Report D676222 (February 1980)

8 Ferguson, Gordon H., Norm A. Cyr and Robert D. Clark, "DIAL: Structural Analysis System User's Manual", LMSC Report (January 1979)

9 Nagarajan, S., "User's Manual for NEPSAP", LMSC Report (November 1978)

10 Loden, William A. and Lawrence E. Stearns, "User's Manual for the REXBAT Program", LMSC Report D460625 (January 1976)

11 Felippa, Carlos A., "Skymatrix Processing Library (SKYPUL) User's Manual", LMSC Report D623146 (January 1978)

12 Gibbs, N.E., W.G. Poole, Jr. and P.K. Stockmeyer, "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix", ICASE Report (July 1974)

STRUCTURAL OPTIMIZATION STUDY

Paul S. Jensen
W. A. Loden

ABSTRACT

An optimization algorithm based on augmented Lagrangians is described. It is implemented with a selection of several popular low rank metric update procedures. Step lengths are determined by line searches using quadratic interpolation in order to avoid excessive gradient evaluations. Penalty weights are formed as simple arithmetic or geometric progressions. Test results and parameter studies are given for well known constrained and unconstrained problems.

TABLE OF CONTENTS

1. INTRODUCTION

This study was conducted as a supplementary effort to a general investigation [1] relating to the optimization of the buckling loads for reinforced structural panels. The original study, reported in [1], included three efforts related to structural optimization, namely:

1. Organize and link appropriate structural analysis computer programs into a system suitable for carrying out structural optimization,

2. Incorporate a convenient optimizer (optimization program) in the system to facilitate the optimization process and

3. Investigate other optimization algorithms that are suitable for structural analysis.

The optimizer used in 2. was CONMIN [6], which is based on a method of feasible directions. CONMIN has been been extensively used for structural optimization and is in a robust, finely tuned state of development. Its primary shortcoming is the fact that the algorthm used does not incorporate any of the innovations developed during the last decade for constrained optimization. Thus, its operational efficiency can probably be improved. In part 3. above, an adaptive algorithm based on the method of augmented Lagrangians was formulated and partially implemented. The implementation of that new algorithm was called ALMIN (Adaptive Lagrange MINimization).

The objective of this effort was to complete the implementation of ALMIN and evaluate it. The algorithms used in ALMIN are briefly discussed in the following section. For a more detailed description, see [1]. Test results and conclusions are presented in Secs. 3 and 4.

2. METHOD

There are three major aspects that contribute to the design of any NLP (Nonlinear Programming) algorithm, namely;

1. The enforcement of the constraint conditions,

2. The selection of step directions for modifying the design variables and

3. Determination of the step magnitudes.

A fourth aspect of key importance to the structural analysis application is the technique used to obtain gradients of the objective and constraint functions. This aspect is not addressed either in the design of ALMIN or the study presented here.

2.1 CONSTRAINTS

We focus on two general techniques for handling constraints. The method of feasible directions, that dates back to the 1950's, attempts to insure that each step (value assigned to the set of design variables) satisfies all of the constraints. The more recent augmented Lagrangian method transforms the constrained optimization problem into a series of unconstrained ones by augmenting the Lagrangian objective function with a penalty function of the constraints. ALMIN (Adaptive Lagrangian MINimizer) uses the latter method.

The augmented Lagrangian approach permits steps that may not satisfy all of the constraints, but produces a penalty value in the augmented objective function that tends to drive subsequent steps back towards the feasible region. The heuristic used produces a mild penalty for the early steps and gradually increases it as the process continues. Unfortunately, if a large number of steps are taken, the objective function can become dominated by the penalty function part of the augmented objective function, occasionally causing convergence to an incorrect result.

There have been some promising developments in the control of the penalty functions recently [2,5], but these have not been incorporated in ALMIN. The heuristic used in ALMIN simply takes an initial scalar penalty weighting factor and multiplies it by a constant (typically 2) each step until it reaches a preset limit.

2.2 STEP DIRECTIONS

In contrast to the complexity of choosing a feasible step direction, as required by the feasible directions method, a simple linearization approach suitable for unconstrained optimization is used for Lagrangian methods. This approach is characterized by the affine approximation

$$g' \cong g + H(x'-x),$$

where the design variable vector is given at two points x and x', and g and g' are the corresponding gradients of the AOF (augmented objective function), see [3] for example. Matrix H is typically an approximation to the Hessian of the AOF. Since the gradient is zero at the optimum point, we have

$$s = (x'-x) \cong H^{-1} g$$

for the choice of step direction.

Since there is no way of producing the Hessian directly for the structural optimization problem, it is approximated by means of the iterates (values of the design variable vector) produced during the optimization process. Starting with H = I, the identity matrix, each successive approximate inverse Hessian is formed by a rank two, variable metric update of the previous approximate. The particular update formulas implemented in ALMIN are:

1. DFP (Davidon-Fletcher-Powell),
2. BFS (Broyden-Fletcher-Shanno),
3. FS (Fletcher switch) and
4. OS (Oren-Spedicato).

It has been found advisable to occasionally reset H = I during the optimization process. The test results in Sec. 3 include parameter studies on the effects of variations in the frequency of resetting H. Emperical results indicate that H should be reset about every 2n iteration steps, where n is the number of design variables.

2.3 STEP LENGTH

Having determined a step direction for modifying the design variable vector, the length of the step to be made must be established. This is determined by searching for a point that minimizes the AOF (augmented objective function) in the chosen direction. Since gradients of the AOF are so costly to obtain in structural optimization, the search method used requires only values of the AOF at intermediate points. Specifically, the rather complex sequence of quadratic interpolation polynomials described in [5] is used for the linear search.

There are controversial opinions regarding how completely the linear searches need to be carried out. Since it is quite expensive and the step direction is only a crude estimate anyway, many authors recommend a cursory linear search. It has been this authors experience that, unless the linear searches are done quite well, the total number of steps required tends to be disproportionately large.

3. TESTS

In order to gain insight into the behavior of the two algorithms (feasible directions and augmented Lagrangians) under study here, a fairly simple constrained optimization problem called the HBR (Hunch Back Rhinoceros) problem was studied in some detail. The name is derived from the shape of the feasible region shown in Fig. 1. The optimum point is at the tip of the "horn" (1,0).

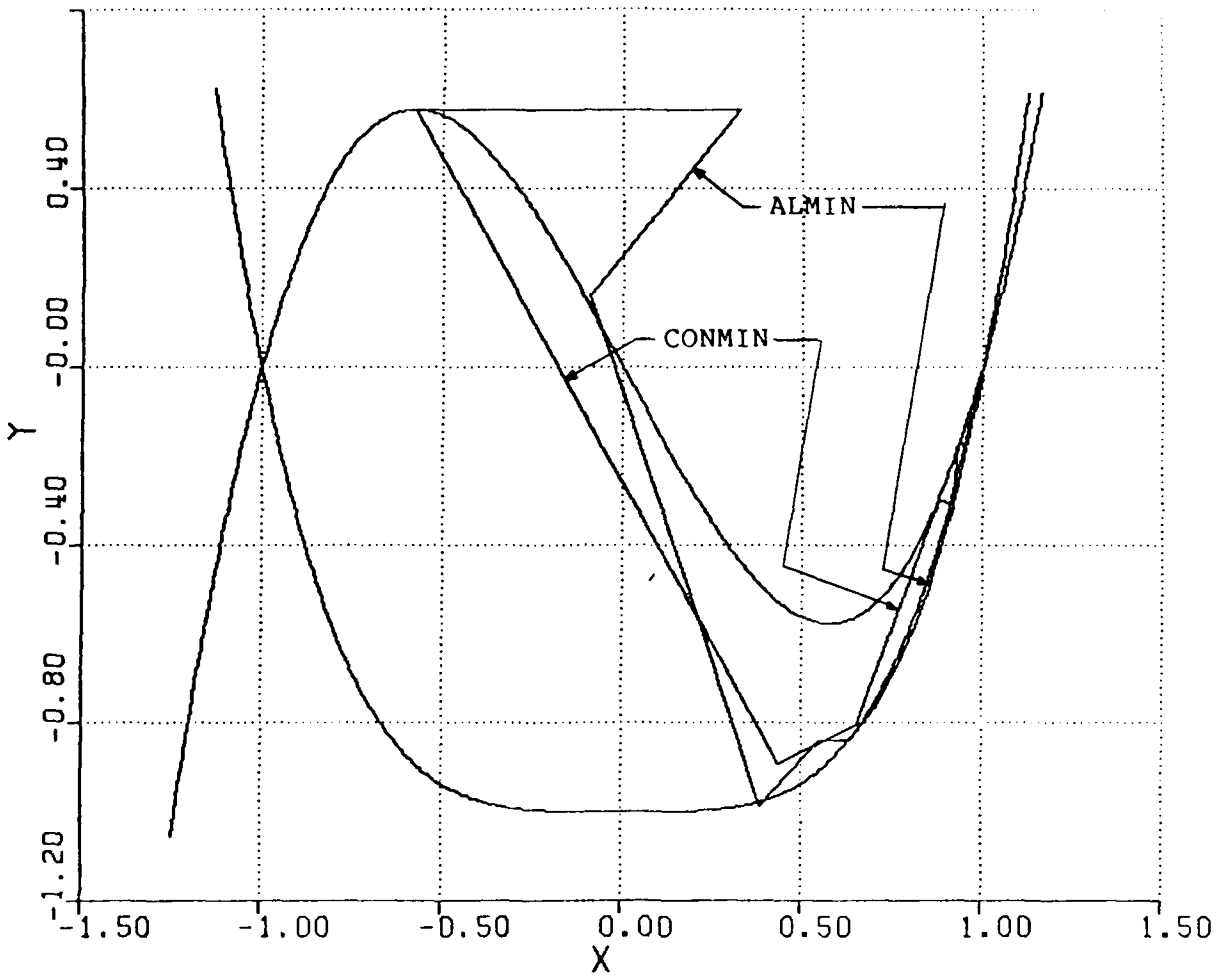


Figure 1. Hunch Back Rhinoceros Problem with Convergence Trajectories

The objective function is

$$f(x,y) = -x$$

with inequality constraints

$$y - x^4 + 1 \geq 0$$

$$1.5x(x^2-1) - y \geq 0$$

The Lagrange multiplier is 1, giving Lagrangian function

$$\ell(x,y) = (x^2-1)(x^2-1.5x+1) - x$$

A Lagrange multiplier of -1/7 produces a stationary point of attraction at the maximum (-1,0), which confused the augmented Lagrangian algorithm until a test for the sign of the multiplier was included to avoid convergence there. The convergence trajectories produced by the two algorithms are also shown in Fig. 1, illustrating behavior typical of many problems.

The remaining test problems used in this study were taken from the broad collection treated by Pierre and Lowe [5]. Specifically, tests were run on the following:

Unconstrained Set

T1: Rosenbrock Banana Function,
T2: Wood Four Variable Function and
T3: Powell's Four Variable Function.

Constrained Set

T7: Pierre's Around the World Problem,
T12: Beale's Three Variable Problem,
T13: Powell's Five Variable Problem and
T17: Rockafellar's Two Variable Problem.

The designators Ti, i = 1,2, ..., are exactly those used in [5].

<u>Unconstrained Problem</u>

The three standard unconstrained problems T1,T2 and T3 were run in order to assess the operation of the code independent of the the penalty function aspect. Convergence was achieved in all three cases with performance somewhat poorer than has appeared in the literature, e.g., [3]. This is possibly due to ALMIN'S special linear search algorithm that avoids the use of gradients.

These tests were run using the DFP update formula and varying the number of linear searches that were carried out between each initialization of the approximate inverse Hessian. The initial Hessian was always taken to be the identity matrix. Specific results for these problems appear in Tables 1 - 3. Entries marked with * did not converge before a preset iteration limit was reached.

The results appear to suggest that the number of searches per initialization should be about twice the number of variables in the problem. Fortunately, however, this does not appear to be a sensitive parameter.

TABLE 3.1 RESULTS FOR T1, ROSENBROCK BANANA FUNCTION

$$\text{Min. } f = 100(y - x^2)^2 + (1 - x)^2$$

Start (-1.2,1)

Run	Searches per Hessian Initialization	Total No. of Searches	Total No. of Function Evaluations
1	2	24	107
2	3	35	175
3	4	36	161
4	5	34	135
5	6	34	138
6	10	36	148

TABLE 3.2 RESULTS FOR T2, WOOD'S FOUR VARIABLE FUNCTION

$$\begin{aligned}\text{Min. } f = {} & 100(x - w^2)^2 + (1 - w)^2 \\ & + 90(z - y^2)^2 + (1 - y)^2 \\ & + 10.1((x - 1)^2 + (z - 1)^2) \\ & + 19.8(x - 1)(z - 1)\end{aligned}$$

Start at -(3,1,3,1)

Run	Searches per Hessian Initialization	Total No. of Searches	Total No. of Function Evaluations
1	2	125	500 *
2	3	145	500 *
3	4	111	500 *
4	5	90	381
5	6	80	323
6	10	76	301

TABLE 3.3 RESULTS FOR T3, POWELL'S FOUR VARIABLE FUNCTION

$$\begin{aligned}\text{Min. } f = {} & (w + 10x)^2 + 5(y - z)^2 \\ & + (x - 2y)^4 + 10(w - z)^4\end{aligned}$$

Start at (3,-1,0,1)

Run	Searches per Hessian Initialization	Total No. of Searches	Total No. of Function Evaluations
1	2	43	150 *
2	3	37	130
3	4	19	66
4	5	18	62
5	6	16	61
6	9	20	71

Constrained Problem

The same parameter studies carried out for the unconstrained problems were applied to the constrained ones. Superimposed on these, however, were a number of experiments with the penalty weight functions.

For Pierre's problem T7, the default heuristic of simply doubling the penalty weight each iteration caused convergence to the wrong solution (see the discussion in Sec. 2.1). The heuristic that was used to generate the results in Table 3.4 was to start with w = .1 and simply increment it by .1 each iteration.

The rest of the problems presented here were solved with no particular difficulty using the default heuristic described above.

TABLE 3.4 RESULTS FOR T7, PIERRE'S AROUND THE WORLD PROBLEM

Max. y subject to $x^2 + y^2 + z^2 = 1$ and

$2y - x \leq 1$

Start at (-.1,-1,.1)

Run	Searches per Hessian Initialization	Total No. of Searches	Total No. of Function Evaluations
1	2	25	81
2	3	19	65
3	4	22	70
4	5	24	80
5	6	23	77
6	10	31	114

TABLE 3.5 RESULTS FOR T12, BEAL'S THREE VARIABLE PROBLEM

Max. f = 8x+6y+4z-2xx-2yy-zz-2xy-2xz-9

Subject to $x+y+2z \leq 3$ and $x, y, z \geq 0$

Start at (.5, .5, .5)

Run	Searches per Hessian Initialization	Total No. of Searches	Total No. of Function Evaluations
1	2	6	19
2	3	5	15
3	4	5	15
4	9	5	15

TABLE 3.6 RESULTS FOR T13, POWELL'S FIVE VARIABLE PROBLEM

Min. $f = vwxyz$

Subject to: $wx - 5yz = 0$

$v^3 + w^3 = -1$ and

$v^2 + w^2 + x^2 + y^2 + z^2 = 10.$

Run	Searches per Hessian Initialization	Total No. of Searches	Total No. of Function Evaluations
1	2	6	14
2	3	6	20
3	4	5	12
4	5	5	11
	. . .		
	10	5	11

TABLE 3.7 RESULTS FOR T17, ROCKAFELLAR'S TWO VARIABLE PROBLEM

Max. $f = y - x^4 - xy$ subject to $y = 0$

Start at (.5, .5)

Run	Searches per Hessian Initialization	Total No. of Searches	Total No. of Function Evaluations
1	2	10	35
2	3	8	29
3	4	10	42
4	5	9	32
5	6	7	24
6	10	10	36

Results for most of these problems were also obtained using the feasible directions algorithm implemented in CONMIN. These are summarized in Table 3.8 with typical results from ALMIN for comparison.

TABLE 3.8 CONMIN RESULTS WITH COMPARATIVE ALMIN RESULTS

Problem	HBR	T1	T2	T3	T7	T12
Function Evaluations						
CONMIN	182	150	234	317	127	127
ALMIN	114	135	301	62	70	15
Gradient Evaluations						
CONMIN	60	27	42	50	39	40
ALMIN	29	34	76	18	22	5

4. CONCLUSIONS

An implementation of ALMIN was completed and a variety of test problems were run. Although operational, the new algorithm is not suitable for production applications in its current form. A number of critical decision processes are presently carried out using ad hoc heuristics that are neither reliable nor always efficient. Examples of these decision processes are: (1) When to reinitialize the approximate Hessian matrix, (2) What technique to use for increasing the penalty function weights and (3) How to approximate the gradient functions.

Our general feeling is that a number of avenues for considerably improving upon the present technique exist and, because of the substantial analysis costs involved, they should be pursued. Some specific areas of interest are:

1. Careful attention to the finite difference techniques used to approximate the gradients of the constraint and objective functions must be given,

2. Innovative utilization of the information available from the structural analyzer (that is not typically used by optimizers) must be found. An example of such information is the eigenvector associated with the buckling load (an eigenvlaue).

3. Incorporation of recent ideas that exploit the properties of the trajectories of penalty and barrier function methods to overcome the problems associated with the penalty weights, see [4] for example.

ACKNOWLEDGEMENTS

The authors express their thanks to Dr. C. A. Felippa for his help in completing the implementation of the ALMIN program. Thanks also go to Mr. Perry Stern for obtaining the CONMIN test results reported here and for numerous suggested improvements to this manuscript.

REFERENCES

1. Almroth, B. O., and P. Stern, "Imperfection Sensitivity of Optimized Structural Panels," AFWAL-TR-80-3128, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, Ohio (1981)

2. Fletcher, R., "An Ideal Penalty Function for Constrained Optimization," J. Inst. Math. Appl. 15 (1975) 319-342

3. Jensen, P.S., "Design and Implementation of a New Descent Algorithm for Local Optimization," Report LMSC-D683306, Lockheed Missiles and Space Company, Inc., (1980) 61 pp.

4. Murray, W. and M.H. Wright, "Projected Lagrangian Methods Based on the Trajectories of Penalty and Barrier Functions," Tech. Rept. SOL 78-23, Stanford University Dept. of Operations Research, (Nov 1978)

5. Pierre, D.A., and M.J.Lowe, Mathematical Programming Via Augmented Lagrangians, Addison-Wesley, Reading, Mass. (1975)

6. Vanderplaats, G.N., "Structural Optimization by Methods of Feasible Directions, "Computers and Structures 3 (1973) 739-755